青少年学编程系列丛书

机器人 Python 青少年编程开发实例

史向东　邓贵勇　著

TurnipBit 既是一个 Python 学习机，
又能带你进入极客世界！

電子工業出版社
Publishing House of Electronics Industry
北京 • BEIJING

内 容 简 介

本书既是介绍 MicroPython 的快速入门书籍，也是以 TurnipBit 为基础进行 MicroPython 实战应用的书籍。

本书以实验的方式进行讲解，只需跟着实验步骤一步一步进行操作，就可以真正实现零基础也能做硬件。在一些重要的知识点处，加上了一些思考内容。这些内容有些是对本知识点的巩固，有些是对本知识点的外延，如果你有能力，则建议尝试着去完成；如果不知道怎么做，也可以试着问问“百度”。在每个实验的最后，都罗列了本实验的知识要点，帮助你理清知识点，掌握编程思想。

本书旨在帮助读者以最短的时间掌握以 TurnipBit 为基础进行 MicroPython 实战应用，希望对有 MicroPython 程序开发需求的读者有所帮助。

图书在版编目（CIP）数据

机器人 Python 青少年编程开发实例 / 史向东，邓贵勇著. —北京：电子工业出版社，2018.3
（青少年学编程系列丛书）
ISBN 978-7-121-33539-6

Ⅰ. ①机… Ⅱ. ①史… ②邓… Ⅲ. ①软件工具－程序设计 Ⅳ. ①TP311.561

中国版本图书馆 CIP 数据核字（2018）第 013644 号

策划编辑：黄爱萍
责任编辑：葛 娜
印 刷：三河市兴达印务有限公司
装 订：三河市兴达印务有限公司
出版发行：电子工业出版社
北京市海淀区万寿路 173 信箱 邮编：100036
开 本：720×1000 1/16 印张：12 字数：202 千字
版 次：2018 年 3 月第 1 版
印 次：2018 年 3 月第 1 次印刷
定 价：59.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：（010）51260888-819，faq@phei.com.cn。

推　荐　序

计算机科学与技术从诞生至今只有区区几十年的时间，与其他传统学科相比，仍处于“婴儿期”，但是它却给人类社会带来了翻天覆地的变化。根据“摩尔”定律，每隔 18 个月，集成电路元器件的集成密度就将翻一番，这就意味着计算机系统的性能将有一次大的提升。从 2000 年至今，我们亲历了互联网、云计算、物联网、大数据、智能硬件技术的大发展，而眼前正在如火如荼地上演着人工智能技术的大爆发。计算机技术为人类开辟了另一个维度的世界——数字世界，而且其中蕴藏了大量的资源和财富。但就创建历史和规模而言，这个世界里的人类可能仍处于原始社会状态，有很多处女地等待着我们去开发和建设，更可能是竞争和拼抢。在未来的社会中，计算机技术更是一项生存竞争的基本技能。

学好计算机技术其实并不容易，尤其是入门，有别于现实自然界的认知方法，它要求学习者具有一定的逻辑思维能力和思维发散能力。因此，早接触计算机技术，越有利于思维能力的锻炼，形成良好的思维习惯。青少年学习计算机技术，往往是在好奇心驱动下开始的，而在面对大量枯燥无味的代码、协议、专业术语时放弃。针对这种情况，本书是一个非常好的选择。本书通过一系列生动有趣的智能硬件制作实例，让初学者在一步步成就感的驱动下，逐渐领会、掌握编程方法和技术。特别是本书选择时下非常流行的 Python 作为编程语言，其具备简捷、易读性和可扩展性等诸多优点，更有利于初学者掌握和使用。同时，本书也是一本关于物联网和智能硬件技术的入门指导书。本书在内容上设计巧妙，能够由浅入深地引导学习者，实操性强，创新与实践相结合，图文并茂，有很多珍贵的设计方案和参考数据。因此，本书不失为一本很好的青少年计算机入门学习指导书，

再配合配套实验设备的使用，将更有利于对技术知识的深入理解和灵活运用。祝广大读者通过本书的学习能早日步入计算机的世界，开发出自己的创新作品。

周　磊

周磊，毕业于浙江大学信电系电路与系统专业，工学博士，九三学社社员，杭州电子科技大学微电子 CAD 研究所副所长，浙江省智慧城市研究中心智能终端研究所副所长，浙江省“151 人才”、浙江省青年科学家。目前主要研究方向为物联网技术和人工智能。获国防科技进步奖三等奖一次、浙江省科技进步奖二等奖两次、浙江省高校科研成果奖两次。

前　言

我从 2013 年开始接触开源硬件。那时，我经常在博客上写学习笔记，最初只是想记录下学习的过程，以后遇到同样的问题，可以随时翻看。后来，越来越多的爱好者和我一起讨论，就索性建了一个 QQ 群，还编辑了一些电子教程。Arduino 和树莓派一直是我钟爱的开源硬件产品。我会用它们来做盒仔机器人、NAS 家庭服务器、电视机顶盒等小手工，也会用它们来做一些简单的开发。2015 年，在朋友的介绍下，我第一次了解到 MicroPython。由于我一直对 Python 比较热衷，所以很快就迷恋上了 MicroPython。从 TPYBoard 开发板入手，简单学习就可以实现各种小创意。2016 年，英国 BBC 面向青少年推出了支持 MicroPython 的 Micro:Bit 开发板。国内也有了类似的 TurnipBit 开发板。于是，我尝试着用这些开源硬件来教我身边的小朋友们学习 MicroPython 和 Python。

学习过一些“积木类”机器人知识以后，一些孩子及其家长往往会有一个困惑，就是不知道接下来再学习什么，以及如何从拖曳式编程过渡到纯代码编程。为此，我意识到 MicroPython 的简单和方便为青少年学习提供了很好的途径。于是，我开始尝试教两个 10 岁左右的小朋友进行 MicroPython 的学习。在学习初期，我们选择和使用了 TurnipBit 开发板，利用 Web 页面的“拼插”编程与代码对比方式进行学习。我发现小朋友可以很快地接受，并能充满乐趣地用代码来进行程序设计。于是，我萌发了写这本书的想法，想让更多的小朋友接触到编程。

根据青少年对数学、物理等基础学科的学习进度，本书从最简单的“点灯”（点亮 LED 灯）开始，选取了 12 个具有代表性的实验。通过实验，不仅将数据类型、逻辑运算、变量、循环、判断、函数等基础的编程知识贯穿其中，便于学习

掌握，而且还培养了青少年编程的兴趣。如果你是正在上小学或者中学的青少年，如果你是一位正在为孩子如何学习编程而发愁的家长，我相信这本书对你会有所帮助，能够引导你或者孩子快速入门计算机编程，了解 MicroPython 甚至 Python 编程语言。

编程学习的准备工作

在开始学习之前，我们需要做一些准备性工作。本书在第 1 章和第 2 章分别介绍了学习前必须要知道的基础知识。比如什么是计算机语言，在本书中用到的 Python 又是一种什么语言等问题，都会在这两章中做出回答。我们还从众多的 MicroPython 开发板中，选出 TurnipBit 开发板作为学习工具，了解该开发板的性能、特点以及使用方法。如果说硬件和必要的基础知识准备过于枯燥的话，那么在第 2 章中将会用一个最简单的小实验——制作“Hello World!”广告牌，带你学习如何使用 TurnipBit。流程图将是建立程序逻辑思维的第一步，如何绘制流程图也将在这一章中讲述。

基础的编程知识

“神奇的计时器”“方便的加法计算器”“会走的机器人”“好玩的掷骰子游戏”……看题目就觉得是很有趣的实验。通过这些实验，你将会逐步学习到二进制计数法、变量及变量的类型、逻辑运算、判断语句和循环语句。至此，你已经掌握了学习 Python 的基础知识，准备工作基本完成，可以再进行一些更深入的学习了。

简单的硬件开发学习

如果说“积木式”机器人的学习离工业设计还有很大距离的话，那么

TurnipBit 会拉近这一距离，让你感觉到似乎能够像程序员一样进行开发了。TurnipBit 自带的无线模块、磁敏传感器、耳机以及各种外置接口（专业术语是 GPIO 接口），可以带你进入硬件开发学习的领域——你将会学习到使用无线模块如何进行通信、使用磁敏传感器如何确认方向、如何利用硬件开发板演奏音乐、如何通过外置接口扩展更多的功能等知识。当然，你还会进一步掌握函数等更多的语言知识。学习完这一部分，你将能够自己制作无线投票器、指南针、储钱罐等有趣的手工作品。

动手实现 DIY 创意

学习完前几章，你是不是已经有了一定的成就感？你是不是已经能够进行一些纯代码编程了？在本书最后两章，我从众多实验中选取了两个具有一定代表性的实验，与你一起从设计、制作、编程到调试，最终完成 DIY 作品。其中一个是带小夜灯的电子时钟，你将会学习到 DS3231 这类时钟模块的使用，还会学习到光敏电阻是如何感知光线的，在光线发生变化时，会自动调整小夜灯是否点亮。另一个是会思考的避障车，你将会学习到超声波传感器的工作原理，学会使用超声波传感器来判断距离，从而调整小车的运动方向，保证让小车不会碰到障碍物。

本书的使用方法

讲到这里，你一定很想现在就开始学习了吧。不急，我最后再说一下本书的一些使用技巧。

- **如何做**：本书以实验的方式进行讲解，你只需要跟着实验步骤一步一步进行操作，就可以真正实现零基础也能做硬件，感受到其中的乐趣。
- **思考**：在一些重要的知识点处，加上了一些思考内容。这些内容有些是对本知识点的巩固，有些是对本知识点的外延，如果你有能力，则建议尝试着去完成；如果不知道怎么做，则可以试着问问“百度”。

- **要点**：在每个实验的最后，都罗列了本实验的知识要点，帮助你理清知识点，掌握内容。

来吧，让我们开始动手创作吧！

致谢

感谢在最开始帮助我们写这本书的所有人，包括在此过程中一开始联系的、讨论知识重点及实验案例的人。他们是邓贵勇、毕吉涛、曾昭智、林静、孙小冬、宿玉青。感谢山东交通职业学院的房华教授、杨瑞老师给予的帮助与支持。

史向东（网名小五义）

2017 年 11 月 27 日于南京

轻松注册成为博文视点社区用户（www.broadview.com.cn），扫码直达本书页面。

- **提交勘误**：您对书中内容的修改意见可在 提交勘误 处提交，若被采纳，将获赠博文视点社区积分（在您购买电子书时，积分可用来抵扣相应金额）。
- **交流互动**：在页面下方 读者评论 处留下您的疑问或观点，与我们和其他读者一同学习交流。

页面入口：http://www.broadview.com.cn/33539

目　　录

第 1 章　打开极客之门

1.1　TurnipBit 是什么

TurnipBit 是以 microbit 为基础，由 TurnipSmart 公司设计的面向青少年编程学习的开发板。该开发板带有蓝牙、加速度传感器、磁敏传感器、耳机插口、按钮以及 5 × 5 LED 点阵等，配有拼插编程工具以及代码编程工具。如果你是一个零基础的“小白”，那么 TurnipBit 能够让你在较短的时间内学会拼插编程；如果你参加过“乐高”等机器人的学习，已经有了编程的基础，那么 TurnipBit 能够帮助你顺利从拼插编程过渡到代码世界，帮助你成为一个真正的“程序员”。

TurnipBit 可以帮助你实现自己的创意，你可以利用 TurnipBit 实现任何酷炫的小发明，无论是智能小车、指南针、机器人还是乐器。你只需要简单的几节课程学习，就可以充分发挥自己的想象力，打开创意空间。

你还可以使用 25 个可显示消息的 LED 灯，画出想显示的图形和文字；可以通过两个可编程按钮，控制游戏操作或者暂停/播放一首音乐；也可以利用加速度传感器、磁敏传感器检测动作并且告知动作进行的方向，同时它也可以通过低功耗蓝牙模块与其他设备或互联网连接。

1.2　从拼插编程开始

拼插编程主要是针对程序设计的初学者或者是年龄较小的爱好者的。拼插编

程平台利用简单的拖曳插件，实现快速的程序设计。初学者能够很快掌握拼插的方法，并实现自己的想法，培养起对编程的兴趣。你可以通过 www.turnipbit.com 网站，进入“开始编程”来体验拼插平台的优势。这里我们来实现一个“Hello World!”程序。

① 进入 TurnipBit 官方主页 http://www.turnipbit.com/，如图 1-1 所示。

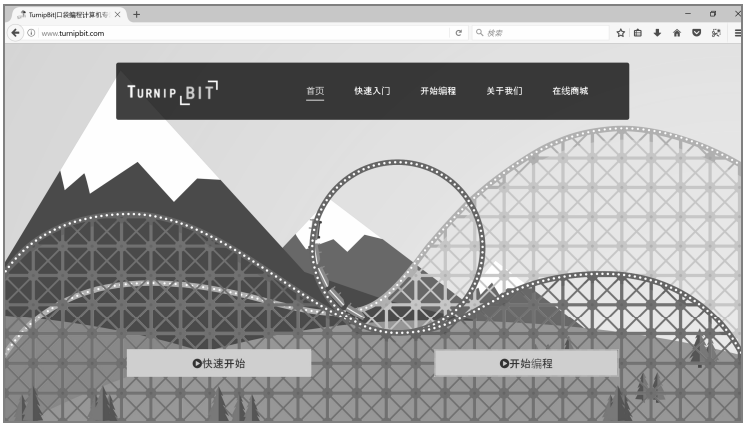


图 1-1　TurnipBit 官方主页

② 点击“开始编程”按钮进入编程界面，进行基础的模块化拼插编程的学习，如图 1-2 所示。

图 1-2　TurnipBit 拼插编程界面

③ 向世界说“Hello World!”。

你只需要编辑一个简单的程序，并将程序下载到名为 TURNIPBIT 的磁盘中，就可以看到程序效果，如图 1-3 所示。

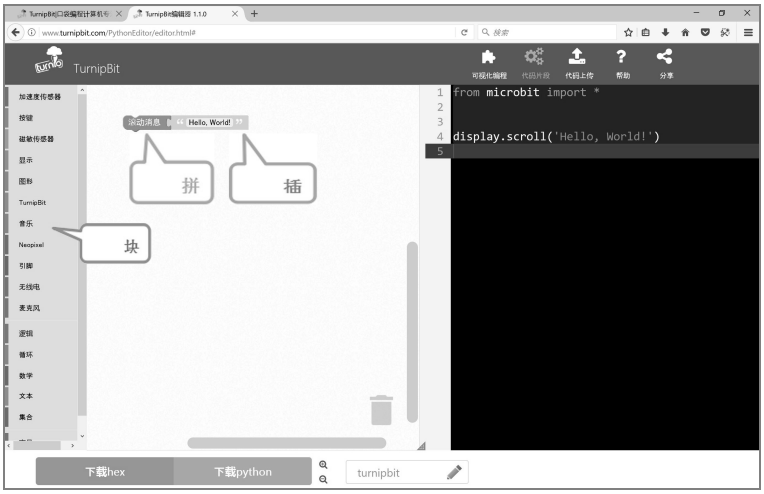


图 1-3　拼插实现“Hello World!”程序

点击“下载 hex”按钮，将程序保存到电脑中，文件的后缀为.hex，如图 1-4 所示。

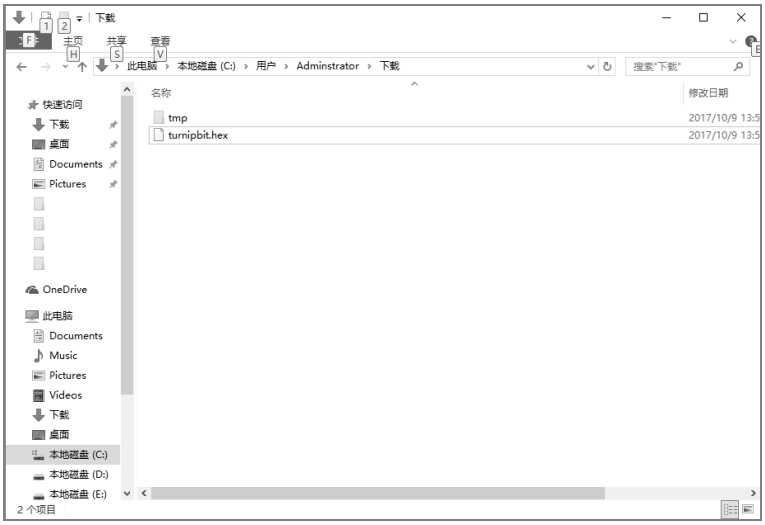


图 1-4　下载 HEX 文件

将 TurnipBit 用下载数据线与电脑连接，就会在电脑中出现名为 TURNIPBIT 的磁盘，如图 1-5 所示。

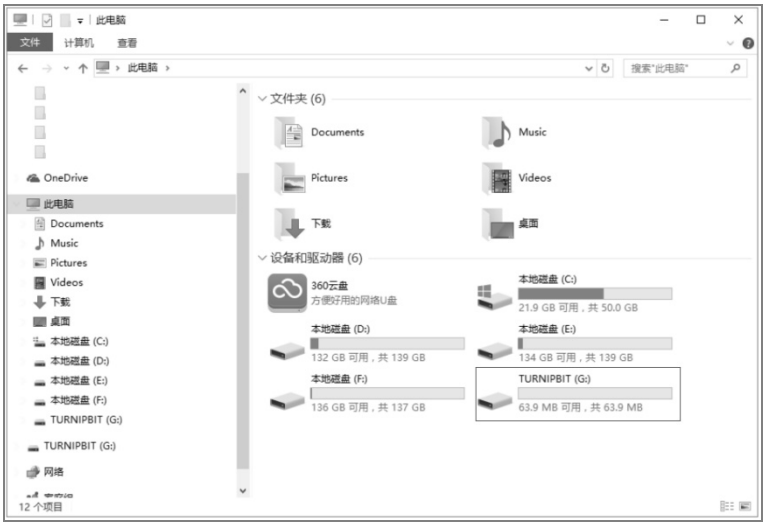


图 1-5　TURNIPBIT 磁盘

将之前保存的 HEX 文件拖入 TURNIPBIT 磁盘中，我们会看到 TurnipBit 板子上的灯在闪烁，说明正在下载到控制板中。下载成功后，会看到程序正常运行。

1.3　做个真正的程序员

拼插编程并不是学习编程的目的，只是一个入门和培养兴趣的途径。要想做个真正的程序员，最终还是要学习计算机语言的。TurnipBit 正是通过拼插与代码的对比实现了快速学习编程的目的。TurnipBit 支持的是 Python 语言，它属于 MicroPython 开发板的一种，其固件中集成了 microbit、music 等多个库文件。

1.3.1　什么是 Python

Python 与 C、Java 等一样，是一门计算机语言，由吉多·范罗苏姆于 1989 年创建。Python 最初是吉多用来打发时间而开发的，经过近 30 年的不断完善，现

在已经成为最受欢迎的语言之一。2017 年 7 月，*IEEE Spectrum* 杂志（美国电气电子工程师学会出版的旗舰杂志）发布的研究报告显示，2016 年排名第三位的 Python，在 2017 年已经成为世界上最受欢迎的语言，C 和 Java 分别位居第二位和第三位。

由于 Python 语言的简洁性、易读性以及可扩展性，一些知名大学已经采用 Python 来教授程序设计课程。例如卡内基梅隆大学的编程基础、麻省理工学院的计算机科学及编程导论就使用 Python 语言讲授。除高校以外，国外在面向青少年编程的教育中也开始引入 Python。Python 在我国也相当流行，各著名高校都开设了 Python 课程，同时一些面向青少年的教育机构、中小学校也积极引入 Python 作为编程启蒙课。

Python 在执行时，首先会将.py 文件中的源代码编译成 Python 的 Byte Code（字节码），然后再由 Python Virtual Machine（Python 虚拟机）来执行这些编译好的字节码。与计算机编程不同的是，在 TurnipBit 开发板的编程中，则是将 Python 代码编译为 HEX 文件，然后烧录到开发板中进行执行。

1.3.2　面向硬件的 MicroPython

2013 年年底，英国剑桥大学数学科学中心理论物理学家 Damien P. George（乔治·达明）提出了 MicroPython。在工作中，他经常使用 Python 语言，并进行一些机器人项目开发。有一天，他突然有了一个想法，能否用 Python 语言来控制单片机，实现对机器人的操控呢？

要知道，Python 虽然很强大，但遗憾的是，它不能实现一些非常底层的操控，所以在硬件领域并不起眼。为了突破这种限制，Damien 花费了 6 个月的时间来打造 MicroPython。MicroPython 基于 ANSI C，语法跟 Python 3 基本一致，拥有自己的解析器、编译器、虚拟机和类库等。目前它支持基于 32bit 的 ARM 处理器，比如 STM32F405、STM32F407 等。借助 MicroPython，用户完全可以通过 Python 脚本语言实现硬件底层的访问和控制，比如控制 LED 灯、LCD 显示器、读取电压、控制电机、访问 SD 卡等。

MicroPython 在支持库方面围绕着 MCU 的特性进行了精减，并形成了 5 个版

本，包括开源的标准版本（standard）、双精度浮点版本（double FP）、支持线程功能版本（threading)、双精度浮点+线程版本（double FP + threading）以及支持网络功能版本（network）。主要的固件结构如表 1-1 所示。

表 1-1　固件组织结构简表

目　　录	解　　释
py/	Python 核心实现，包含编译器和 Runtime
unix/	UNIX 版本的 MicroPython
stmhal/	运行在 STM32F405RG 上的 MicroPython 接口和底层驱动
unix-cpy/	输出字节码的 MicroPython
tests/	测试框架和测试脚本
tools/	dfu 工具
logo/	Logo
mpy-cross/	Mpy 编程工具

1.3.3　支持 MicroPython 的开发板

MicroPython 从诞生的那一刻起，就被很多人关注，相应的开发板也应运而生。国际上最有名的就是 Pyboard，有 PYB v1.0 和 PYB v1.1 等多款。在我国 TPYBoard 是做得比较成熟的，涵盖系列较多。本书中使用的 TurnipBit 与 TPYBoard 相似，是支持 MicroPython 的开发板之一。

1．Pyboard 的兴起

Damien 利用 STM32F405 制作的第一块支持 MicroPython 的开发板就是 Pyboard。它基于 STM32F405 单片机，通过 USB 接口进行数据传输。该开发板内置 4 个 LED 灯、一个加速度传感器、时钟模块，可在 3～10V 的电压下正常工作。值得一提的是，它遵守 MIT 协议开源，被授权人拥有复制、修改、发行和再授权的权利。到目前为止，Pyboard 已经有两个版本，分别是 v1.0 和 v1.1。

MicroPython 一经公开，并在 Kickstarter 上进行众筹后，就引起了各地技术人员的关注，在极短的时间内，MicroPython 被移植到了 ESP8266、ESP32、STM32F407 等开发板上。

2．我国的 MicroPython 开发板

随着 MicroPython 被广泛关注，我国的许多技术人员也开始仿照 Pyboard 制作自己的开发板，如制作了 Pymagic、PYB nano 等。当前，影响力较大的当属 TPYBoard。这块开发板与 Pyboard 功能类似，也采用了 STM32F405 作为 MCU（微控制单元，又称单片机），具有 4 个 LED 灯以及一个加速度传感器。该开发板除支持 DFU 外，还加入了 SWD 接口，可利用该接口进行固件调试。TPYBoard 发布后，也一度引起了国外技术人员的关注，在 MicroPython 的官方论坛上进行了多次讨论。

目前 TPYBoard 已经发展出多个版本，包括 STM32 标准版、基于 LAN 的有线版、基于 ESP8266 的无线版等，并发布了自己的固件。

（1）STM32 标准版

该版本包括支持 TPYBoard 固件及兼容 PYBv10 的 V102 开发板（见图 1-6）和支持 TPYBoard 固件及兼容 PYBv11 的 V103 开发板。其中 V103 为高端板，采用黑色镀金工艺，除适合实验学习外，也可应用在工业设计中。

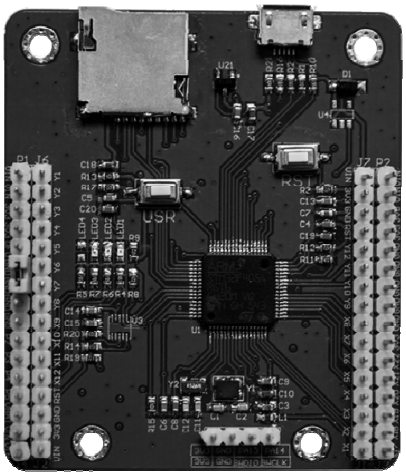


图 1-6　TPYBoard V102 开发板

（2）基于网络的 TPYBoard V20X 系列

V20X 为 TPYBoard 的网络系列，更适合物联网使用。其中包括基于 LAN 的

V201 开发板（见图 1-7）和基于 Wi-Fi 的 ESP8266 开发板 V202（见图 1-8）。V20X 开发板的应用更加广泛，可轻松实现 Socket 服务器、Web 服务器，也可以完成 Socket、TCP 以及 UDP 通信。

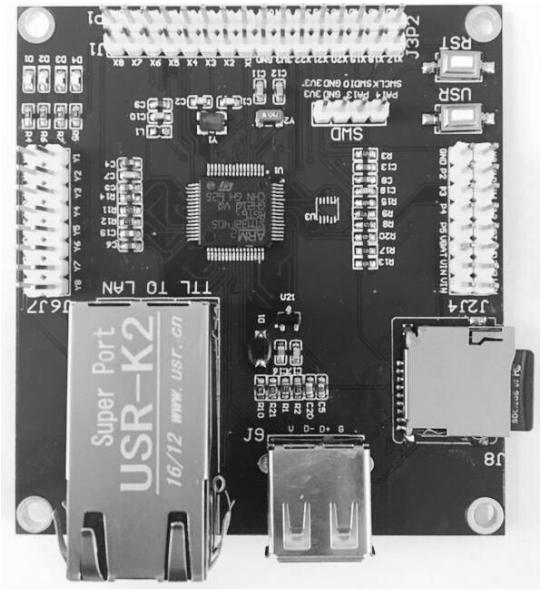


图 1-7　TPYBoard V201 开发板

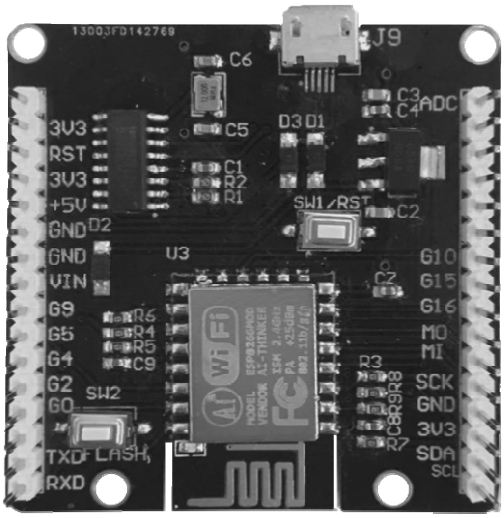


图 1-8　TPYBoard V202 开发板

（3）基于北斗定位的 TPYBoard V70X

TPYBoard V70X 系列开发板（见图 1-9）集成了 GU620 芯片，该芯片支持北斗定位或者 GPS、GPRS 基站定位，使得 TPYBoard 可以直接利用 MicroPython 进行 GPS 定位开发。同时 TPYBoard V70X 还自带一套 Web 接收与控制服务器，使用者可以轻松完成定位硬件及 Web 端展示。

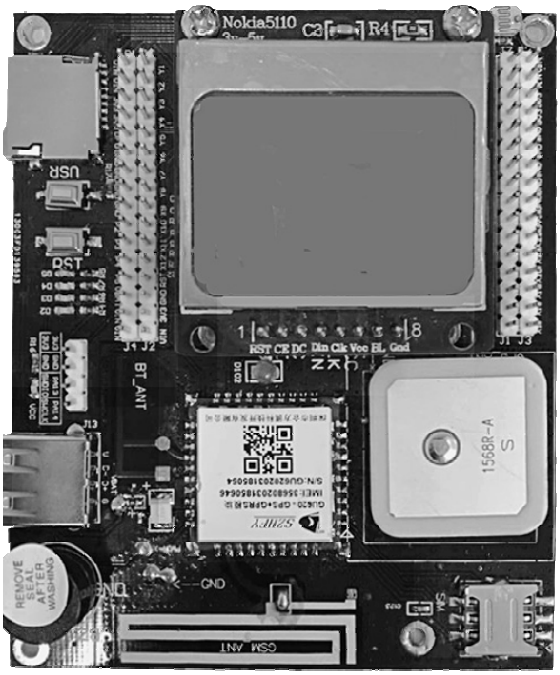


图 1-9　TPYBoard V70X 系列开发板

1.3.4　利用 TurnipBit 进行编程学习

随着信息技术的发展与普及，越来越多的家庭开始重视青少年编程学习，以编程能力为基础的信息奥林匹克竞赛和 STEM 教育等一度受到追捧。TurnipBit 正是在这种背景下产生的。TurnipBit 支持 MicroPython，将易接受的拼插编程与纯代码编程相结合，帮助青少年快速从拼插编程过渡到代码编程，建立程序员思维。通过拼插与代码的对比学习，青少年很容易理解和掌握变量、数学运算、逻辑运算、循环、函数等基本知识，对于进行 Python 学习培养兴趣，建立基础。

TurnipBit 的正面和背面示意图分别如图 1-10 和图 1-11 所示。

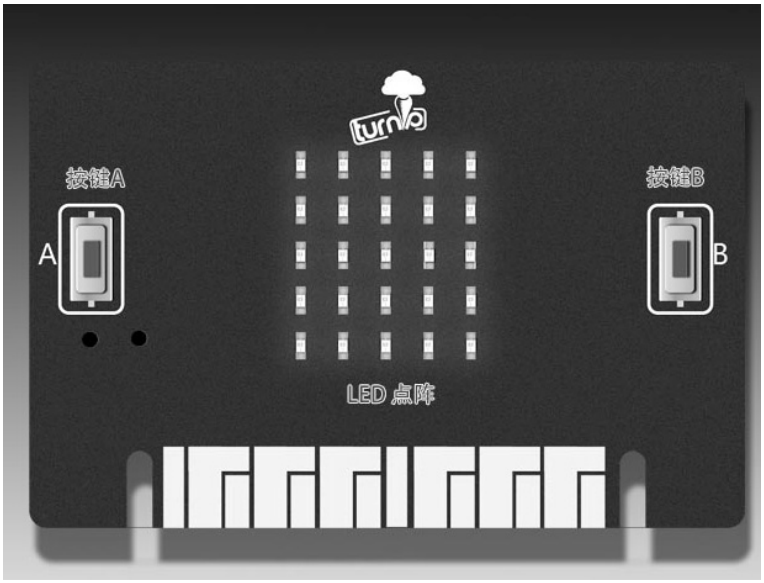


图 1-10　TurnipBit 正面示意图

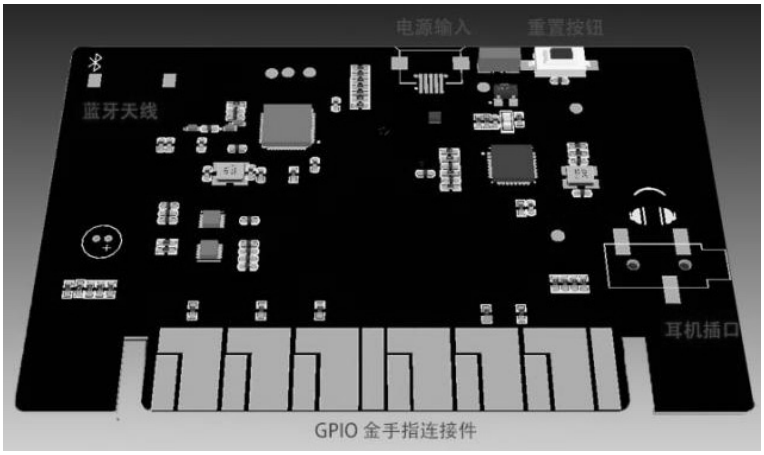


图 1-11　TurnipBit 背面示意图

TurnipBit 的基本硬件操作包括以下几个部分。

（1）电源输入，TurnipBit 电源输入采用 microusb 接口，电压为 5V，其位置如图 1-11 所示。

（2）重置按键（见图 1-11），即我们常说的 RESET 键，用来重启开发板。

（3）按键 A 和 B（见图 1-10），即两个用来进行控制的按键，在进行程序设计时，我们可以利用该按键完成指定任务。

（4）耳机接口（见图 1-11），用来通过耳机输出声音。

（5）GPIO 金手指连接件（见图 1-11），该连接件将 GPIO 从开发板中引出，实现扩展功能。例如，我们可以利用所引出的引脚接 DS3231 时钟模块，也可以用来接温度传感器等其他模块。各引脚的具体定义如图 1-12 所示。

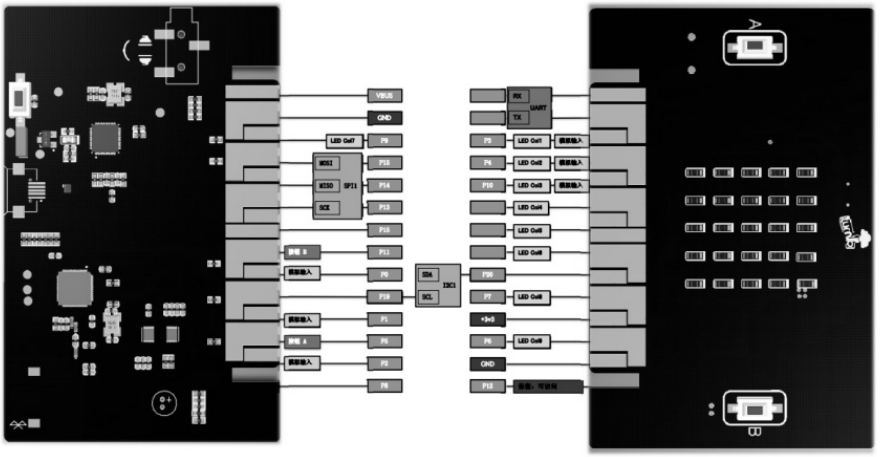


图 1-12　TurnipBit 引脚图

接下来，让我们一起从广告牌制作、倒计时器、计算器、会走的机器人、无线投票器等实验开始奇妙的编程之旅吧！

第 2 章　滚动的广告牌

2.1　滚动的“Hello World!”

滚动字幕的广告牌在城市的大街小巷经常能够看到，这种广告牌在白天和夜间都可以进行信息展示。

Hello World!的中文意思是“世界，你好！”。因 *The C Programme Language* 中使用它作为第一个演示程序而变得非常有名，所以后来程序员在学习编程或进行设备调试时都习惯使用“Hello World!”来作为第一个程序。现在我们就利用 TurnipBit 板载的 5×5 LED 点阵来完成滚动显示字符“Hello World!”，标志 MicroPython 学习之旅的开始。

所需器材：

- TurnipBit 开发板一块
- 下载数据线一条
- 接入互联网的电脑一台（推荐使用 Google Chrome 或者 Firefox 浏览器）

2.2　动手进行拼插编程

2.2.1　实现滚动显示“Hello World!”

① 打开官方网站 http://www.turnipbit.com/（如图 2-1 所示），点击“开始编程”

按钮进入编程界面，开始学习基础的模块化拼插编程，如图 2-2 所示。

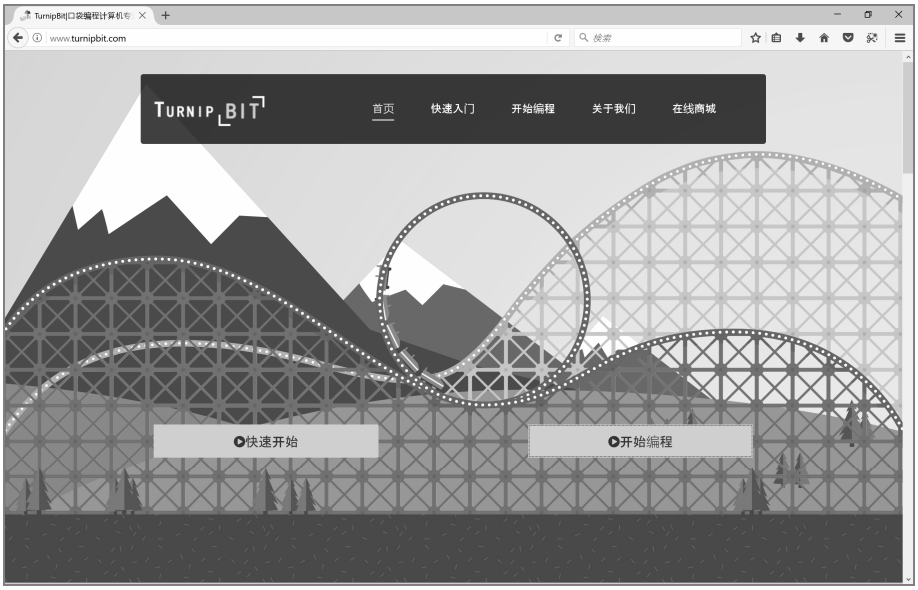


图 2-1　TurnipBit 官方主页

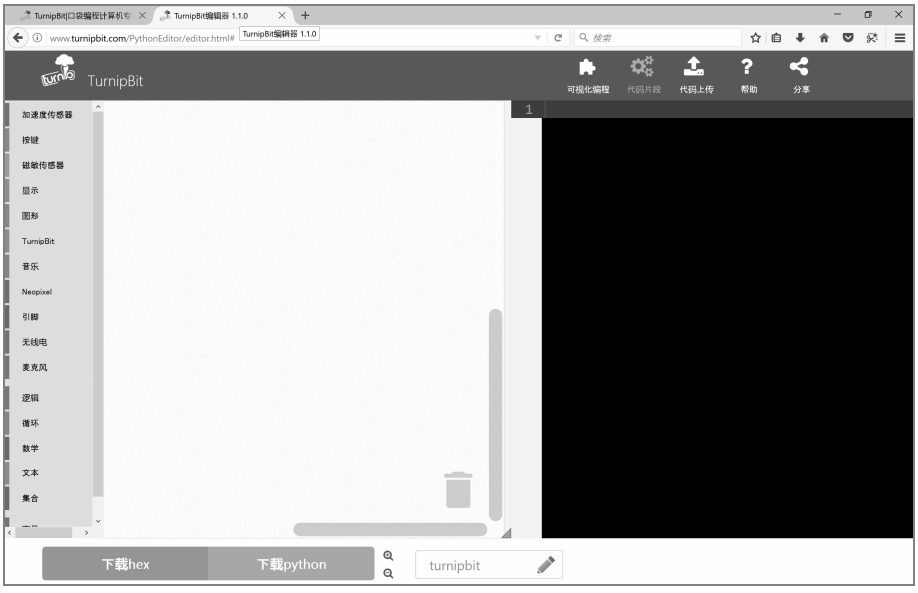


图 2-2　TurnipBit 编程界面

② 在左侧的块选择区找到“显示”块，用鼠标选中 

，如图 2-3 所示。

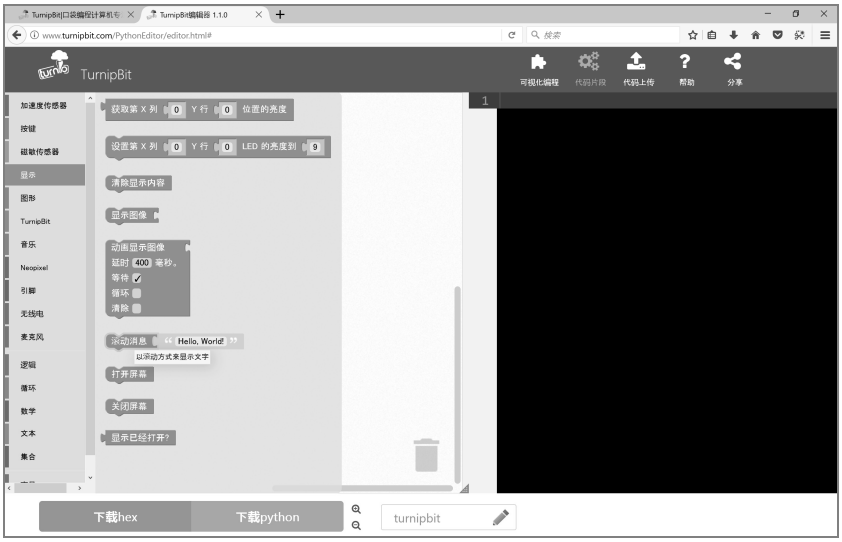


图 2-3　使用“显示”块

拖曳至模块编辑区，如图 2-4 所示。

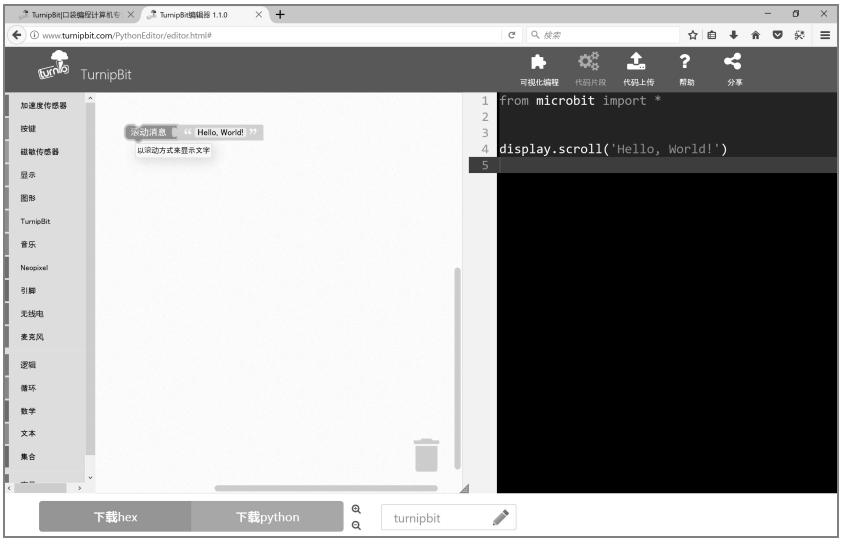


图 2-4　拖曳“显示”块

③ 保持程序名 TurnipBit 不变，点击“下载 hex”按钮将程序保存到电脑中，如图 2-5 所示。

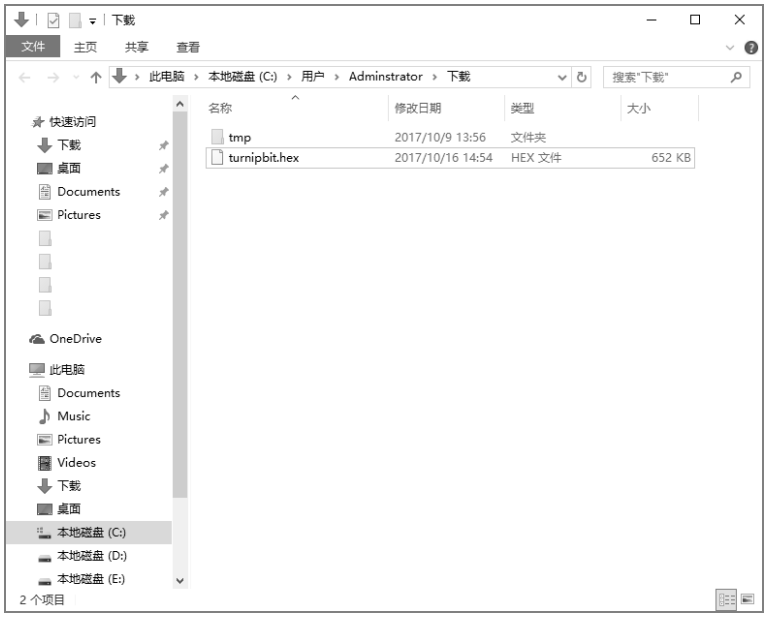


图 2-5　保存 HEX 文件

把所保存的 HEX 文件拖入 TURNIPBIT 磁盘中，我们会看到 TurnipBit 板子上的灯在闪烁，说明正在下载到控制板中。下载成功后，会看到程序正常运行，如图 2-6 所示。

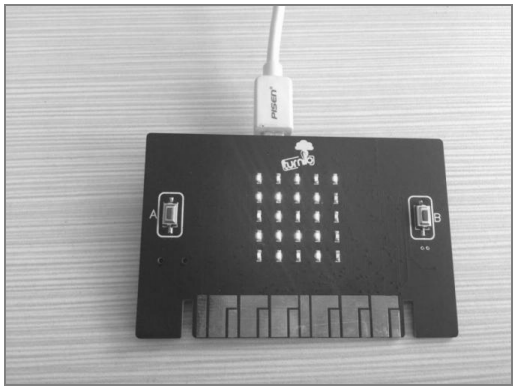


图 2-6　TurnipBit 运行“Hello World!”

【思考】

细心的同学会发现滚动的“Hello World!”只显示了一次，而实际生活中的 LED 广告牌是如何显示的呢？

比如按一下“重置”按键看看会发生什么情况呢？

那么如何制作出可以循环滚动显示的 LED 广告牌呢？

2.2.2 实现循环滚动显示“Hello World!”

① 在左侧的块选择区找到“循环”块，用鼠标选中  ，拖曳至模块编辑区，如图 2-7 所示。

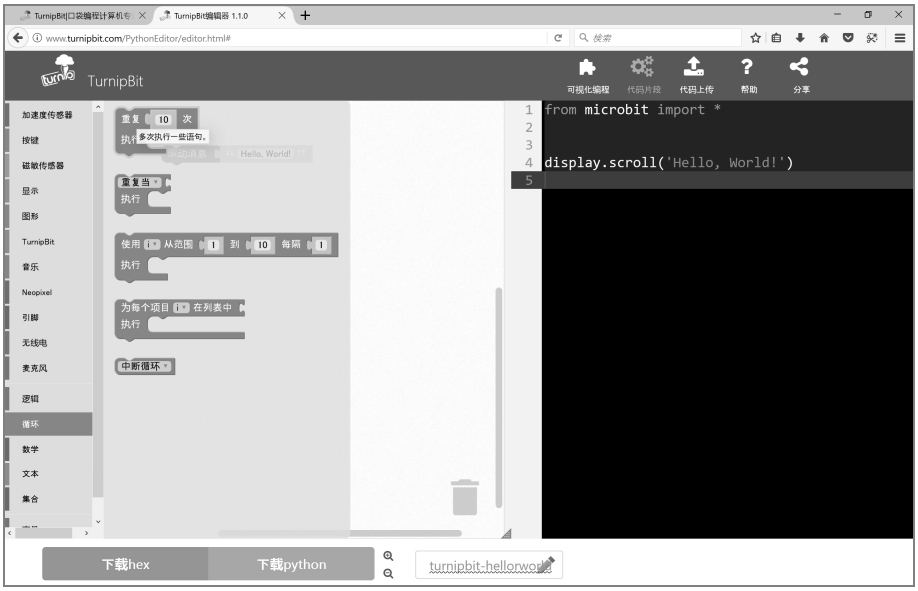


图 2-7　使用“循环”块

② 把“滚动消息”拖到“重复 10 次”中，表示循环 10 次，如图 2-8 所示。

③ 保持程序名 TurnipBit 不变，点击“下载 hex”按钮将程序保存到电脑中，如图 2-9 所示。

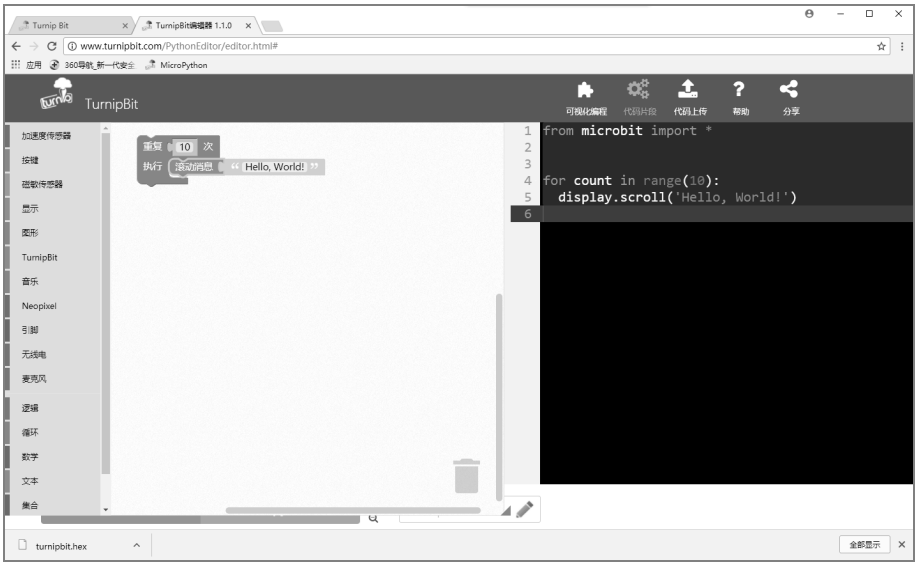


图 2-8　循环 10 次

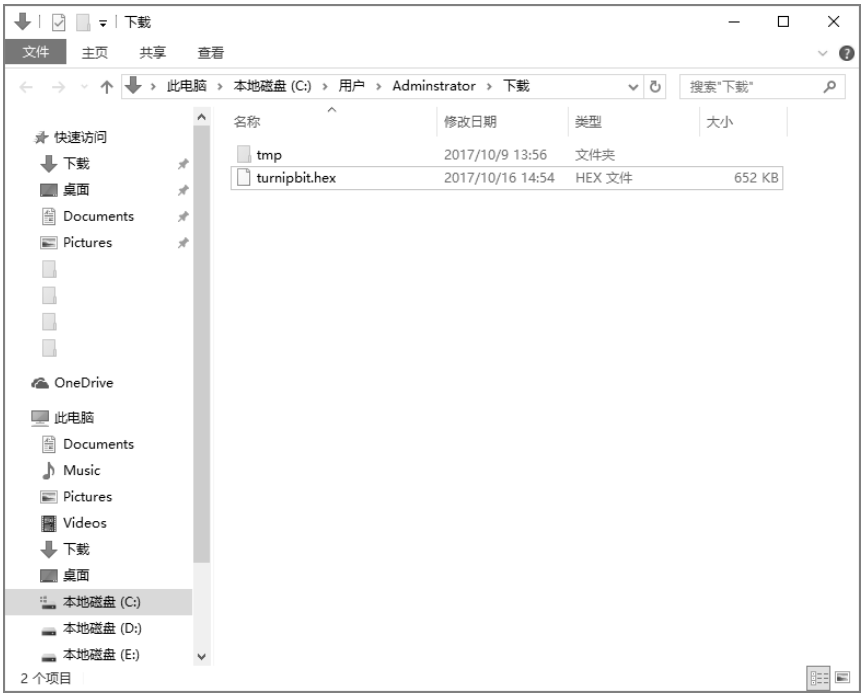


图 2-9　保存 HEX 文件

把所保存的 HEX 文件拖入 TURNIPBIT 磁盘中，我们会看到 TurnipBit 板子上的灯在闪烁，说明正在下载到控制板中。下载成功后，会看到程序正常运行。

【思考】

发现有什么变化了吗？

有没有一直进行循环显示的方法？这个问题留给同学们自己研究。

2.3 动手画流程图

2.3.1 流程图是什么

程序员在进行程序设计前或者设计中，为了便于梳理自己的思路往往会绘制流程图。流程图（Flow Chart）是使用图形表示算法的思路的一种极好的方法，千言万语不如一个图。

我们一起看看下面的例子，如图 2-10 所示。

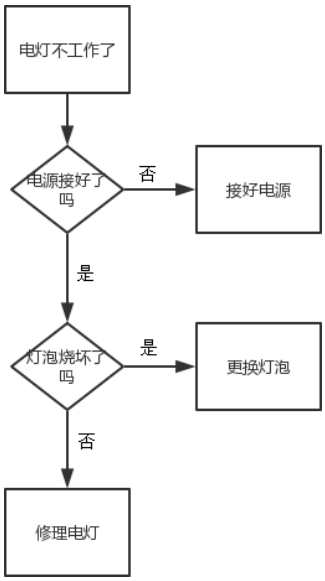


图 2-10　流程图示例

从图 2-10 来看，这个流程图是用来发现和解决电灯故障的。当电灯出现故障时，首先看有没有接通电源，如果没有接通就进入“接好电源”的流程；如果电源已经接通，那么就进一步检查“灯泡烧坏了吗”，如果烧坏了就“更换灯泡”；如果不是灯泡烧坏了，那么就“修理电灯”。现在，我们基本就把这个程序的思路梳理清楚了，于是开始编程。

在进行程序设计之初，我们首先会根据流程图写出伪代码（Pseudocode）。伪代码是一种算法描述语言。使用伪代码的目的是使被描述的算法可以很容易地以任何一种编程语言（如 Pascal、C、Java 等）来实现。因此，伪代码必须结构清晰、代码简单、可读性好，并且类似于自然语言。在发现和解决电灯故障的例子中，我们将使用流程图所梳理出来的思路用伪代码加以表现，具体如下：

```
Begin（算法开始）
如果（if） “电源没接好”：
    那么 “接好电源”
否则：
    如果（if）“灯泡烧坏了”：
        那么“更换灯泡”
    否则：
        “修理电灯”
End（算法结束）
```

2.3.2　画出“Hello World!”的流程图

对于“Hello World!”这个程序来说，其思路比较简单，就是在循环内“滚动显示 Hello World!”，所以可以用伪代码与流程图表现如下。

“Hello World!”程序的伪代码：

```
Begin（算法开始）
循环（For） 变量=初值（0） To 终值（10） 步长 1
执行滚动消息
End（算法结束）
```

“Hello World!”程序的流程图如图 2-11 所示。

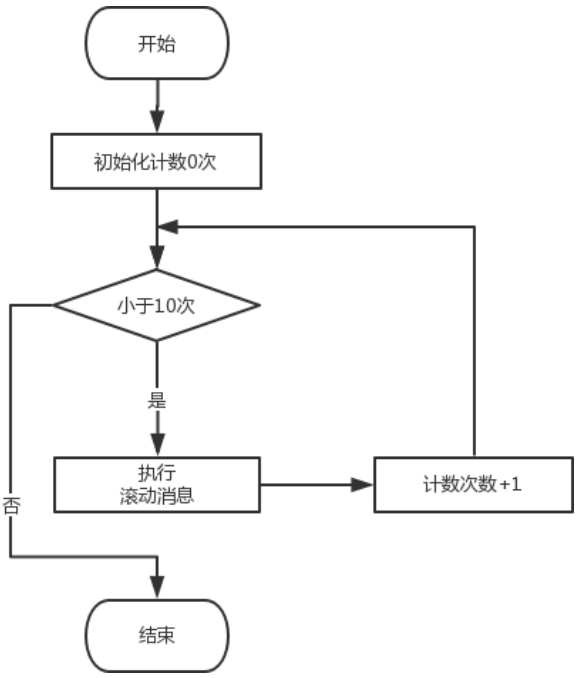


图 2-11 “Hello World!”程序的流程图

【思考】

没有循环时怎么画流程图？

2.4 知识要点

2.4.1 拼插编程

【掌握】

- TurnipBit 拼插编程网站的代码切换。
- 会将程序下载到口袋编程计算机 TurnipBit。
- 会使用“滚动消息”拼。
- 会使用“重复 10 次”拼。

【理解】

- 拼插编程中的“块”“拼”“插”。
- For 循环的应用。
- “循环”拼在编程中的应用

2.4.2　代码编程

【掌握】

- 流程图中的“开始/结束”的用法。
- 流程图中的“过程”的用法。
- 流程图中的“判定”的用法。

【理解】

- 什么是流程图？
- For 循环中的隐含计数和判断。

第 3 章　倒计时

3.1　神奇的计时器

简单地说，计时器就是用来记录时间的。当前计时器主要分为两种，一种是正向的计时器，往往从 1 开始，计数到 9999 等，时间间隔固定。比如在一些比耐力和持续性的项目中，常常用这种计时器来记录流逝的时间。另一种是倒向的计时器，又称倒计时器，它往往从一个固定的值（如 9）倒向减到 0，标志着开始。比如在重要赛事、重要活动的开始或者一些电影的片头，都会进行 10 秒的倒计时。2008 年，北京奥运会开幕式的倒计时使用的就是倒计时器，从 10 减到 0，烟花齐放，奥运会开幕。

现在，我们用 TurnipBit 来做一个 9 秒的倒计时器，从 9 开始，每隔 1 秒减少 1，最终减少到 0。

所需器材：

- TurnipBit 开发板一块
- 下载数据线一条
- 接入互联网的电脑一台（推荐使用 Google Chrome 或者 Firefox 浏览器）

3.2　让 TurnipBit 显示数字

3.2.1　实现滚动显示数字

① 打开官方网站 http://www.turnipbit.com/，点击“开始编程”按钮进入编程界面，进行基础的模块化拼插编程的学习，如图 3-1 所示。

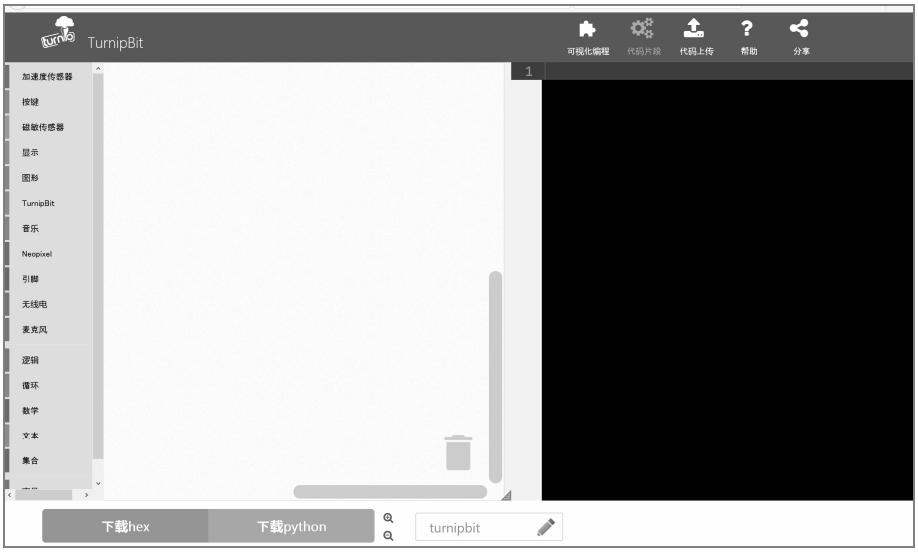


图 3-1　TurnipBit 编程界面

② 还是使用第 2 章中提到的“滚动消息”，只是把“Hello World!”换成数字，如 

。首先将其拖曳至模块编辑区，双击“Hello World!”，然后改为数字 1，如图 3-2 所示。

③ 将程序名修改成“TurnipBit-numb”，点击“下载 hex”按钮将程序保存到电脑中，如图 3-3 所示。

把所保存的 HEX 文件拖入 TURNIPBIT 磁盘中，我们会看到 TurnipBit 板子上的灯在闪烁，说明正在下载到控制板中。下载成功后，会看到 TurnipBit 的 LED 屏会有一个数字 1 滚动而过，如图 3-4 所示。

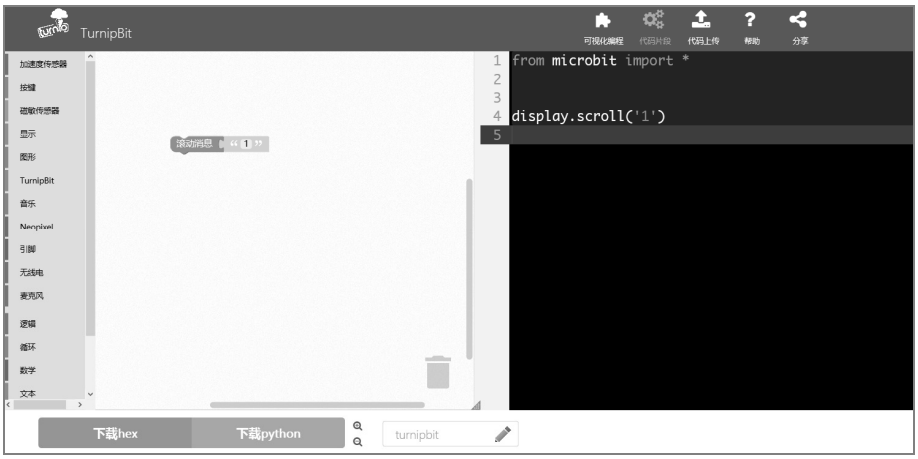


图 3-2　滚动显示数字 1

图 3-3　保存 HEX 文件

图 3-4　显示数字实验图

3.2.2　显示静态数字

上一节中，我们实现了滚动显示数字，数字从 LED 屏滚动过后，所有的 LED 灯就熄灭了。那么，能不能在 LED 屏上显示一个静态的数字呢？这里我们用拼插和代码两种方法来实现，TurnipBit 可以帮助你从拼插向专业代码进行转变。

方法 1：以画图方式实现显示静态数字。

① 从块选择区找到“显示”块，并将“显示图像”拖动到模块编辑区，然后从“图形”块中找到“创建图像”并拼插在“显示图像”的后面，最后在“创建图像”中绘出数字 1，如图 3-5 所示。

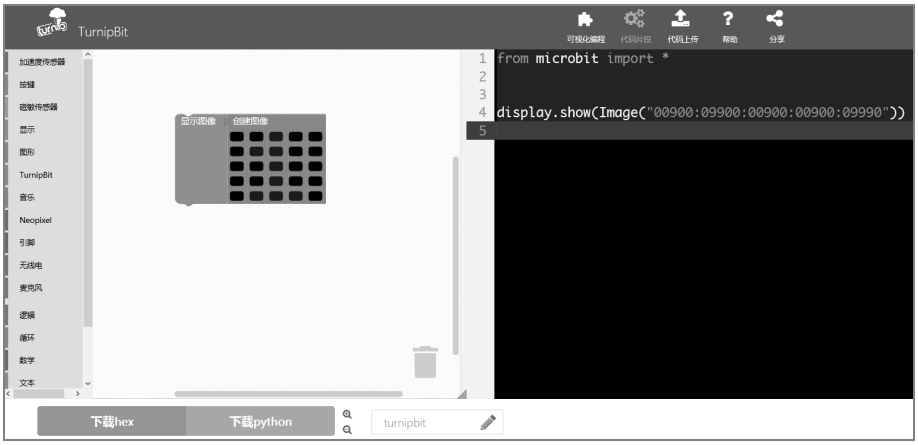


图 3-5　绘出数字 1

② 点击“下载 hex”按钮，保存 HEX 文件，然后将该文件拖入 TURNIPBIT 磁盘中，我们会看到 TurnipBit 板子上的灯静态显示数字 1，如图 3-6 所示。

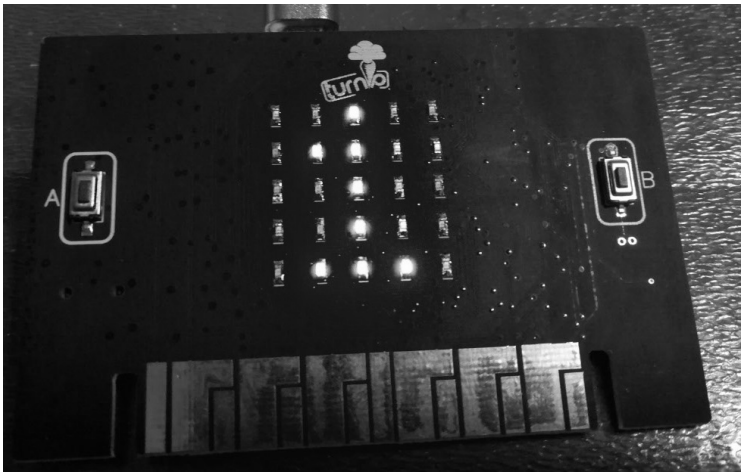


图 3-6　静态显示数字 1

【思考】

如果每个数字都用画的方法，是不是会很慢呢？

有没有更方便、更专业的方法呢？

方法 2：用代码实现静态显示数字。

细心的同学会发现，在拼插模块的同时，右边的代码显示区会显示对应的代码，前面滚动数字、画图的显示都对应了不同的代码。例如：

“滚动数字”对应的拼插件为“滚动显示‘1’”，代码为 display.scroll("1")；

“静态数字”对应的拼插件为“显示图像-创建图像”，代码为 display.show (Image("00900:09900:00900:00900:09990"))。在这行代码中，每组 00000 代表每一行有 5 个 LED 灯，如第一组 00900 表示第一行第三个灯亮，亮度为 9。这里亮度共分 10 个等级，0 为最暗，即不亮；1~9 为逐渐增亮。

那么，能不能直接通过代码来静态显示数字呢？通过上面的代码我们会发现，两行代码中有一个共用的模块，即 display（显现）。在 Python 语言中，常见的语言规范为：

```
[[模块.]/[对象.]].函数名（）
```

这里的 display 就可以视为模块，其后面跟的 scroll()或者 show()就是函数。可以看出，show()显示了静态内容，而 scroll()显示了动态内容。在 show()函数中，如果在括号内填入字符（字符是一种数值类型，字母或者数字加引号后就变成了字符），开发板就会显示出来。比如 display.show("A")，开发板上的 LED 灯就会显示一个大写的 A，这样做是不是比拼插更专业、更简单？

具体步骤如下：

点击右上角的“可视化编程”，切换到代码编辑模式，如图 3-7 所示。

在代码显示区，在“from microbit import *”的下面写入“display.show("1")”，如图 3-8 所示。

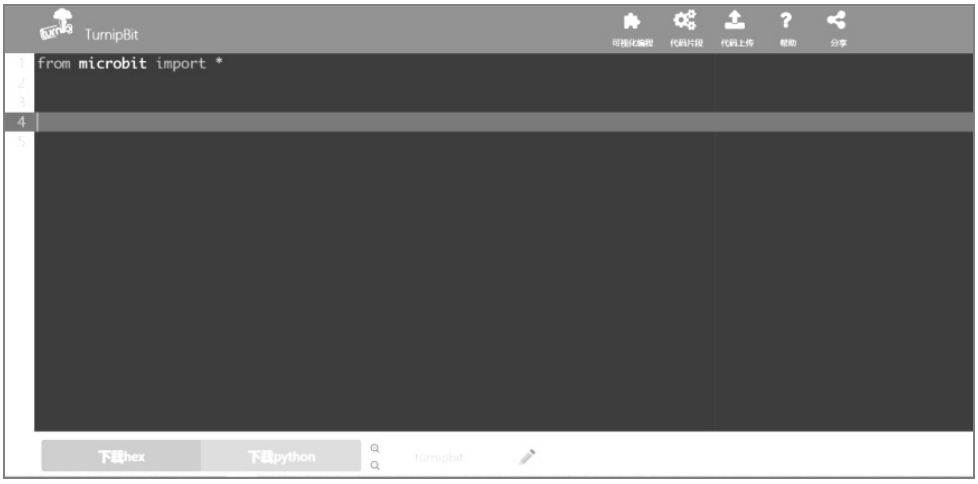


图 3-7　代码编辑模式

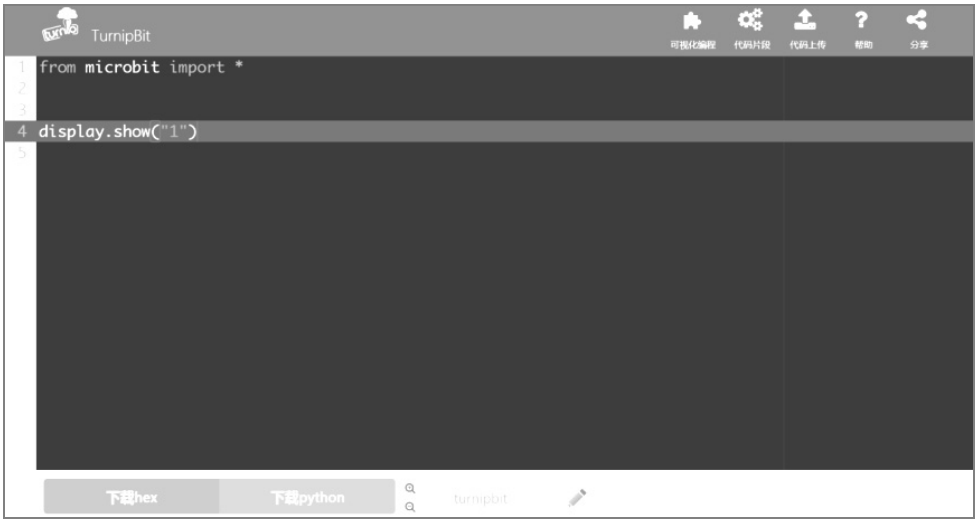


图 3-8　通过代码显示“1”

点击“下载 hex”按钮，保存文件，并将该 HEX 文件拖到 TURNIPBIT 磁盘内，此时 TurnipBit 显示数字“1”，与前面拼插编程实现的效果是一样的，如图 3-9 所示。

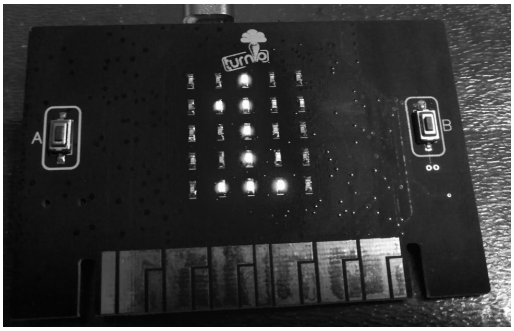


图 3-9　静态显示数字“1”

【思考】

用代码的方法是不是更简单？现在你能自己做倒计时器了吗？

3.2.3　有趣的数字

至此，我们学习了显示文本（“Hello World!”）和数字。但不知道细心的你有没有发现，在显示数字的时候，数字都是用双引号引起来的，这是什么意思呢？其实在 Python 语言中，加了双引号的数字表示的是字符，如“5”并不是表示数字 5，而是表示字符 5。引号告诉计算机当前这个数字是字符。如果多个字符连接在一起，就形成了字符串，如“Hello World”“123456”。

```
>>>num="5"  #表示字符 5
>>>numb=5  #表示数字 5
>>>first_str="i am first"   #字符串
```

关于字符串我们会在后面的章节中进行讲述，这里先来看一下计算机中的数字。通常来说，数字包括 0、1、2、3、4、5、6、7、8、9，以及由这些数字组成的组合，下面我们就来认识这些神奇的数字。

1．0 是什么

0 是什么？“今天的销售额为 0”，这里的 0 表示什么也没有。0 就是表示什么也没有吗？其实不然，在不同的使用场景下，0 表示的意义有所不同。例如：

温度 0℃并不是表示没有温度，而是表示比-1℃高，比 1℃低。

“为增加收入，接点零活”，这里的零也不是表示没有活。

在电子技术中，0 一般表示低电平，1 一般表示高电平，在 TurnipBit 开发板中也是这样设置的。在逻辑运算中，0 表示逻辑假（False），1 表示逻辑真（True）。在 Python 语言程序设计中，未赋值的变量（详见第 4 章）为 0；0 的位置代表不同的数据类型，如 10 表示整数 10，而 1.0 并不是指整数 1，而是指浮点数 1.0。

2．认识二进制

我们使用的是十进制形式，而计算机使用的却是二进制形式来保存数据和编写程序的。为什么计算机用二进制形式而不用十进制形式呢？用十进制形式不是更方便吗？

如果要让计算机用十进制形式来保存数据，那么首先需要计算机能够识别十进制里的 0~9 十个数字。怎么识别呢？通常来说，可以通过元器件的电压高低水平来进行衡量。例如，电路最高电压为 12V，那么将 10 个数字均分到 0~12V 之间，每个数字的电压间隔为 1.33V（如图 3-10 所示）。如此小的电压区间，要准确测量出电压，并标记为数字，是相对比较复杂的。

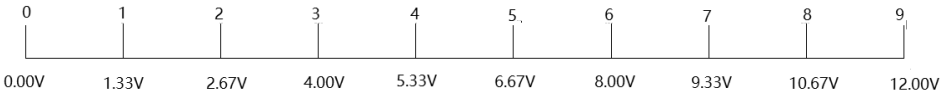


图 3-10　电压分布图

相比来说，使用二进制形式就简单和方便得多，因为具有两种稳定状态的元器件很容易被找到，如晶体管的导通与截止、继电器的接通与断开，以及电脉冲的高低电平等。技术实现简单、运算规则简单、逻辑运算方便、易于转换等优势，是二进制形式在计算机中应用的主要原因。二进制计数与十进制类似，只有 0 和 1 两个数码，且满 2 进 1。十进制数由个位、十位、百位、千位、万位等组成，二进制数则由 1 位、2 位、4 位、8 位、16 位、32 位等组成（如表 3-1 所示）。

表 3-1　进制位数表

进制	1	0	0	0	0	0	0	0
十进制	千万位	百万位	十万位	万位	千位	百位	十位	个位
二进制	128 位	64 位	32 位	16 位	8 位	4 位	2 位	1 位

3．二进制运算规则

与十进制相似，二进制运算也包括加、减、乘、除基本运算。具体方法如下。

（1）加法

二进制加法中加数和被加数只能为 0 或 1，因此只有四种情况。

```
0+0=0
1+0=1
0+1=1
1+1=10 #进 1 位
```

根据加法交换律，第二种和第三种是一种情况，所以实际上二进制加法只有三种情况。多位数的二进制数相加也和十进制类似。例如：

```
101+1001=1110
```

用竖式表示，如图 3-11 所示。

```
  1 0 1
+ 1 0 0 1
---------
1 1 1 0
```

图 3-11　二进制加法竖式计算

（2）减法

减法的规则也很简单，只有四种情况。

```
0-0=0
1-0=1
1-1=0
0-1=1 #借 1 位当 2
```

例如，计算 1110 减去 1001：

```
1110-1001=101
```

用竖式表示，如图 3-12 所示。

```
  1 1 1 0
- 1 0 0 1
---------
0 1 0 1
```

图 3-12　二进制减法竖式计算

（3）乘法

二进制乘法的规则如下：

```
0×0=0
1×0=0
1×1=1
0×1=0
```

多位数的二进制乘法就是在这个规则下进行的，如计算 1001×101，其竖式表示如图 3-13 所示。从竖式来看，当乘数某位为 0 时，与被乘数相乘结果全为 0；当乘数某位为 1 时，与被乘数相乘得到的是被乘数对应的各位，只是需要将被乘数向左移动相应的位数，于是二进制乘法就可以简单地转化为“加法与移位”。

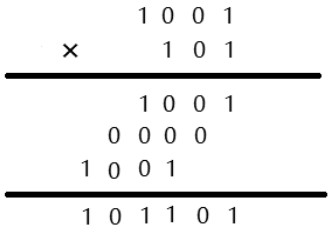


图 3-13　二进制乘法竖式计算

（4）除法

二进制除法除去 0 作为除数无意义外，只有两种情况。

```
0÷1=0
1÷1=1
```

多位数的二进制除法，如 110011 除以 11，其竖式表示如图 3-14 所示。从竖式可以看出，二进制除法实际上就是“减法与移位”。

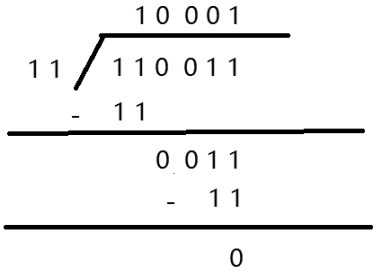


图 3-14　二进制除法竖式计算

4．进制转换

（1）二进制转换为十进制

在二进制的计数规则中，从右到左依次为 1 位、2 位、4 位、8 位……每位有不同的数字，即 0 或者 1。每位的 0 或者 1 分别表示不同位的数字个数。比如二进制数 110，右边的 1 位的 0 表示的是 1 的个数，2 位的 1 表示的是 2 的个数，4 位的 1 表示的是 4 的个数。于是 110 就变成了 1×0+2×1+4×1=6。因为二进制的位数是 2 的幂次关系，所以二进制转换为十进制就可以用 2 的幂来计算，比如：

$$110 = 1\times 2^2 + 1\times 2^1 + 0\times 2^0 = 4 + 2 + 0 = 6$$

再比如：

$$110110 = 1\times 2^5 + 1\times 2^4 + 0\times 2^3 + 1\times 2^2 + 1\times 2^1 + 0\times 2^0 = 32 + 16 + 0 + 4 + 2 + 0 = 54$$

（2）十进制转换为二进制

与二进制转换为十进制相反，十进制转为二进制通常使用“除 2 取余、逆序排列”法。具体做法是：用 2 整除十进制整数，可以得到一个商和余数；再用 2 去除商，又得到一个商和余数，依次进行，直到商为 0。将先得到的余数放在最右边作为最低位，后得到的依次放在左边作为高位，就得到了二进制结果。比如将上面的十进制数 54 转换为二进制形式，如图 3-15 所示。于是将 54 转换为 110110。

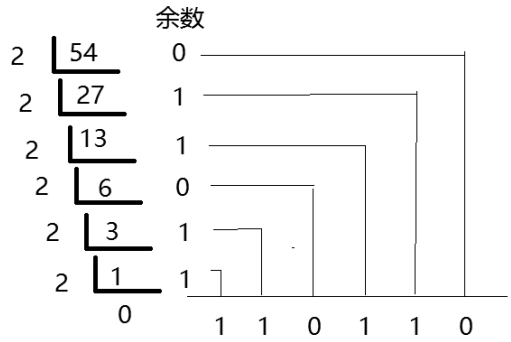


图 3-15　将十进制数 54 转换为二进制形式

3.3　动手制作倒计时器

3.3.1　“倒计时器”程序流程图

让我们先用伪代码来分析倒计时器的制作逻辑。此倒计时器为从 9 倒计时至 0，每个数字间隔 1 秒。通过分析，需要每隔 1 秒显示一个数字，且数字从 9 减少到 0。于是，我们会考虑首先显示 9，然后停 1 秒，显示 8，再停 1 秒，显示 7……直到显示 0。逻辑顺序如下：

第 1 步，显示 9；

第 2 步，停 1 秒；

第 3 步，显示 8；

第 4 步，停 1 秒；

……

第 *n* 步，显示 0；

画出流程图，如图 3-16 所示。

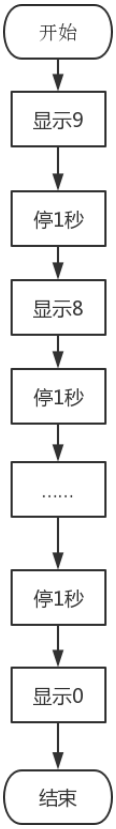


图 3-16　“倒计时器”程序流程图

3.3.2　睡眠 1000 毫秒

从上面的流程图来看，“显示数字”是刚刚学过的，但是“停 1 秒”怎么实现呢？于是，我们回到“可视化编程”中，在块选择区选择“TurnipBit”中的“睡眠 1000 毫秒”，如图 3-17 所示。

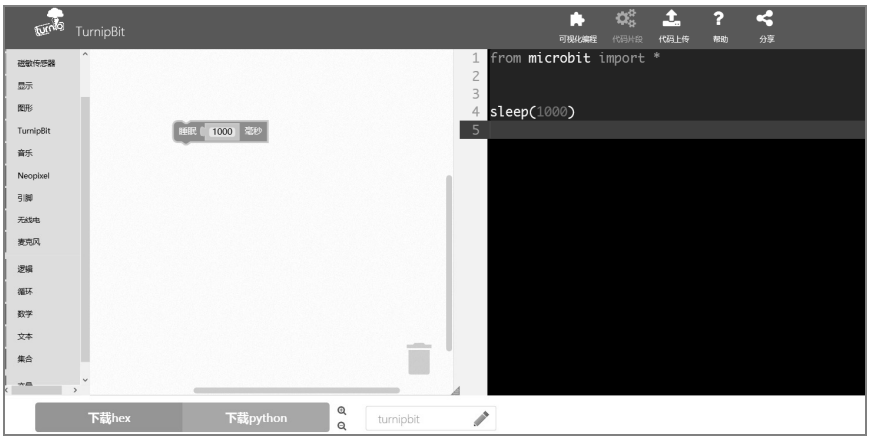


图 3-17　选择“睡眠 1000 毫秒”

讲到这里，我们需要学习一下时间单位的换算。1 小时等于 60 分钟，1 分钟等于 60 秒。那么，什么是毫秒呢？毫秒是比秒更短的时间单位，1 秒可以等分为 1000 毫秒。因此，“睡眠 1000 毫秒”其实就是“停 1 秒”，对应的代码为 sleep(1000)。

3.3.3　完成“倒计时器”

掌握了上面的知识，我们就可以用程序来制作一个“倒计时器”了。点击“可视化编程”，进入编程模式，根据流程图在代码显示区写入代码，如图 3-18 所示。

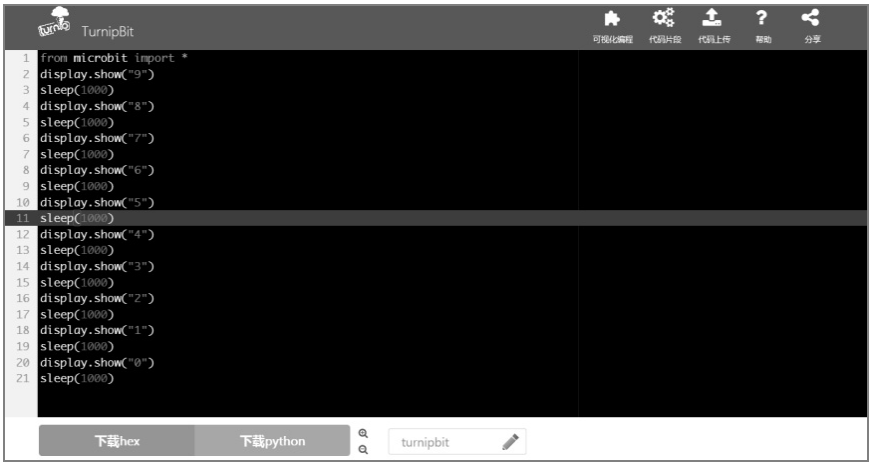


图 3-18　“倒计时器”程序

点击“下载 hex”按钮保存文件，并将该文件拖到 TURNIPBIT 磁盘中，可以看到 TurnipBit 从 9 开始倒计数，直到 0。

【思考】

从 9 到 0，每秒减少 1，是不是很有规律？能不能按照这个规律写一个更简单的程序呢？等后面学习了变量和循环后，大家可以自己尝试。

3.4　知识要点

3.4.1　拼插编程

【掌握】

- TurnipBit 滚动显示数字和静态显示数字的几种方法。
- 用代码的形式实现静态显示数字。
- 二进制运算规则。
- 二进制与十进制的转换。

【理解】

数值类型中的字符与数字的区别。

3.4.2　代码编程

【掌握】

- display 模块的使用方法，以及 display.show()与 display.scroll()的用法。
- sleep()的用法。

第 4 章　方便的加法计算器

4.1　DIY 加法计算器

每天那么多的数学作业，要是有个计算器该多么方便啊！在我们的日常生活中，计算器再常见不过了，几乎每家都有计算器。那么，计算器是怎么做出来的呢？我们能不能 DIY 一个？

现在，我们就利用 TurnipBit 来完成一个简单的加法计算器，让它来帮助你完成复杂的加法计算。当然，学完这一章，你也完全可以做出一个加、减、乘、除都有的计算器。

所需器材：

- TurnipBit 开发板一块
- 下载数据线一条
- 接入互联网的电脑一台（推荐使用 Google Chrome 或者 Firefox 浏览器）

4.2　变量及其类型

4.2.1　变量

变量是表示可变状态、具有存储空间的抽象。这样的定义，大家看起来会发晕。什么是抽象啊？其实，说白了，变量就是会变的量值。在程序运行过程中，

变量会随着程序的运行而随时改变，可以是一个值，也可以是一组值。

要理解变量，首先要知道“内存”。计算机的存储主要分两个部分，一部分是硬盘。硬盘可以长期存储文件，如电影、游戏等，这种存储不依赖于有没有通电，电脑有没有开机，文件都会一直保存在电脑硬盘里。另一部分是内存。内存只有在通电时才会存储，断电后，内存里的数据就会全部被清空。变量就存储在内存中。

为了更好地理解什么是内存，我们可以把内存看成众多小房间。当电脑通电工作时每个房间存储着不同的东西，当电脑关机后，小房间里的东西就清空了。举个例子来说明，我们学习时，需要用到钢笔、铅笔、橡皮、直尺等。我来管理学生的学习用品，在 1 号房间放钢笔，2 号房间放铅笔，3 号房间放橡皮，4 号房间放直尺。你来找我领学习用品，我需要先告诉你每个房间放的是什么，这个过程叫作“声明”和“赋值”。然后你会去 1 号房间领钢笔。过了几天，我又进了一批量角器，于是我把 1 号房间改放量角器，5 号房间放钢笔，于是再次进行了“赋值”。这时，你又要领钢笔，你就会直接去 5 号房间了。

例子里的每个房间就是变量，只是我们在计算机编程时，一般不用中文，所以不会写成 1 号房间、2 号房间等，而是常用变量名来表示，如用 n 表示 1 号房间，用 m 表示 2 号房间等。那么从“声明”到“赋值”的过程可以表示如下。

声明：

```
n 学习用品类
m 学习用品类
```

赋值：

```
n=pen（钢笔）
m=pencil（铅笔）
```

几天后，再次赋值：

```
n=protractor（量角器）
```

到这里，我们就能理解程序中的变量通常是怎么工作的。例如：

```
i=1
j=5
j=j+i
```

在 Python 语言中，i=1 和 j=5 既是赋值语句又是声明语句，i 赋值为 1，又由

于 1 是整数，所以 i 同时被声明为整数类型（在 4.2.3 节将详细讲解）。当然，为了使程序更加清晰，在编程时也完全可以在程序开始的时候先声明变量类型，如添加 int i,j 声明语句。j=j+i，再次改变了 j 的值，最终 j 不是等于 5，而是等于 6。

【思考】

每个变量都具有类型和值两个属性，那么除可以给变量赋值为整数以外，还能给变量赋什么值呢？你能举出几个例子吗？

4.2.2 变量的命名

每个变量都有一个名字，比如上节中提到的 n、m、i 和 j 等。那么在程序中一般如何给变量起名字呢？

1. 变量命名规范

Python 语言与其他语言一样，对变量命名有一个规范，不是任意的符号都可以用来做变量名的。具体规范为：

- 变量名仅可以包括字母、数字和下画线，且不能以数字开头。比如 id2 可以作为变量名，但是 2id 就不行。
- 系统关键字不能作为变量名使用。什么是系统关键字呢？最简单的例子就是 Python 中固定的关键字。例如，在前面章节的代码中，均以“from microbit import *”开始，它的意思是导入 microbit 库。这里的 from 和 import 就是系统关键字。再比如 for、while、true、int 等均为系统关键字。
- 变量名的长度没有限制，但区分大小写。比如 name 和 Name 就是不同的两个变量。

2. 变量命名方法

变量命名方法有很多，包括匈牙利命名法、骆驼命名法和帕斯卡命名法等。不管哪种方法，只需要做到命名简洁、易懂就可以了。比如可以用 name 来命名“姓名”，用 age 来命名“年龄”等。

【思考】

以下哪些名称可以作为变量名使用？

A. 2name　B. number_1　C. from　D. name_nub　E. True　F. _age

4.2.3　变量的类型

Python 的变量类型主要包括：字符串、布尔类型、整数、浮点数、列表、元组、字典等。

（1）字符串：由数字、字母、下画线组成的一串字符。例如：

```
str='this is string'  #str 就是一个字符串变量
```

（2）布尔类型：用于进行判断或比较数据，只有 True 和 False 两个值。例如：

```
bool=True
```

（3）整数：表示物体个数的数。例如：

```
n=5
```

（4）浮点数：就是小数。例如：

```
fn=5.1
```

（5）列表：Python 中最基本的数据结构。列表中的每一个元素都会被分配一个数字，用来表示它的位置或索引，其第一个索引是 0，第二个索引是 1，依此类推。

```
list1 = ['physics', 'chemistry', 1997, 2000];
```

list1 就是一个列表，其中：

```
list1[0]='physics'
list1[1]='chemistry'
list1[2]=1997
list1[3]=2000
```

（6）元组：与列表类似，不同之处在于元组的元素不能修改。例如：

```
tup2 = (1, 2, 3, 4, 5 )
```

注意：元组是用圆括号括起来的，而列表使用的是方括号。

（7）字典：一种可变的容器模型，且可存储任意类型的对象。字典的键 key

与值 value 之间用冒号（:）分隔，每个键值对之间用逗号（,）分隔，整个字典包括在花括号（{}）中。例如：

```
dict = {'Alice': '2341', 'Beth': '9102', 'Cecil': '3258'}
```

则：

```
dict['Alice']=2341
dict['Beth']=9102
```

另外，Python 中还有日期和时间类型等，这里就不做详细说明了。

【思考】

以下哪个是元组？

A．chartlist = ["a", "b", "c", "d"];

B．numtup= (1, 2, 3, 4, 5);

C．Timelist='20170201'

D．dict2 = { 'abc': 123, 98.6: 37 };

4.2.4 数据类型操作

1．常用的字符串操作

（1）去除空格及特殊符号

```
s.strip()#去除两端的空格
s.lstrip()#去除左边的空格
s.rstrip()#去除右边的空格
```

s 为字符串变量。

（2）连接字符串

```
s1="123"
s2="456"
```

s1+s2 为“123456”。

另外，join()命令也可以实现字符串连接。

（3）查找字符串

```
s.find(str1)# 从 s 中查找 str1 第一次出现的位置，计算机在进行计数时是从 0 开始
```

```
的，所以字符串中的第一个字符的起始计数不是 1，而是 0
    s,rfind(str1)# 从 s 右边最后一位开始查找 str1 第一次出现的位置，返回负值
```

（4）计算字符串长度

```
    len(s)# 计算字符串长度，如 len("hello")的结果为 5
```

（5）截取字符串

```
    s="hello"#s[1]是 e，s[0]是 h，s[0:3]是 he
```

2．常用的列表操作

（1）插入元素

```
    list=['1','2','3']
    list.insert(1,'4')
```

结果：list 变为['1','4','2','3']。

（2）删除元素

```
    list.del(n)# 删除 list 列表的第 n 个元素
    list.remove(str)#  删除 list 中 str 值的元素
```

（3）计算列表长度

```
    len(list)
```

（4）列表排序

```
    list.sort()# 正序排列
    list.reverse()# 反序排列
```

4.3　动手制作加法计算器

4.3.1　加法计算器流程图

通过加法计算器让 TurnipBit 计算两个整数的和。我们先用伪代码来实现。

① 定义变量 sumb，并计算两个数的和赋给 sumb。假设计算 23+12，那么 sumb=23+12。

② 利用 LED 点阵显示加法等式及结果，即显示 23+12=sumb 的值。

如图 4-1 所示为加法计算器流程图。

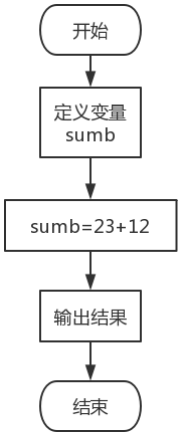


图 4-1　加法计算器流程图

4.3.2　加法计算器的实现

1．拼插实现

在拼插实现的过程中，在最后显示结果时使用滚动显示数字的方法。具体步骤为：

① 定义变量 sumb。点击块选择区中的“变量”，选择“创建变量”，在输入框中输入“sumb”，如图 4-2 所示。

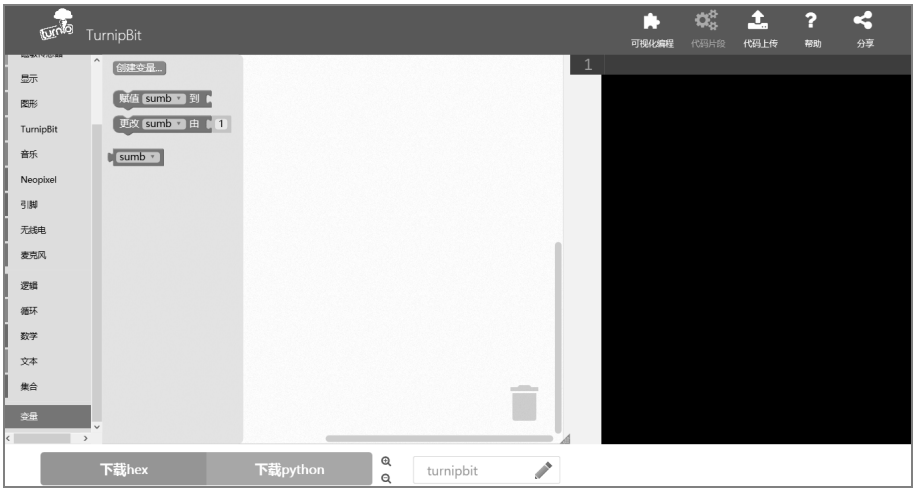


图 4-2　拼插创建变量

② 选择“赋值 sumb 到”给 sumb 赋值。由于 sumb=23+12，所以从块选择区中选择“数学”下的“1+1”，拼插在“赋值 sumb 到”，此时代码显示区出现“sumb=23+12”，如图 4-3 所示。

图 4-3　给 sumb 赋值

③ 显示等式。点击“显示”→“滚动消息‘Hello World!’”，拼插到“赋值 sumb 到”的下面，同时将“Hello World!”修改为“23+12=”，如图 4-4 所示。

图 4-4　显示等式

④ 将计算结果 sumb 显示出来。在这一步，就会用到变量的类型。sumb 是几个整数的和，所以它是整数类型。但是在“滚动显示”模块中，我们会发现“滚动显示”中显示的内容总是在引号里面，这说明“滚动显示”只能显示字符串。所以，我们需要将 sumb 转换为字符串才能正常用“滚动显示”来显示。具体方法是：选用“滚动显示‘Hello World!’”，然后选择“文本”→“建立字符串使用”拼插在“滚动显示”的后面（覆盖“Hello World!”），点击“建立字符串使用”左边的小齿轮图标，从弹出的框右边区域将下面的“项目”删除（由于只有一个显示项目“sumb”，所以只保留一个“项目”即可），如图 4-5 所示。最后再点击小齿轮图标，选择“变量”→“sumb”，如图 4-6 所示。

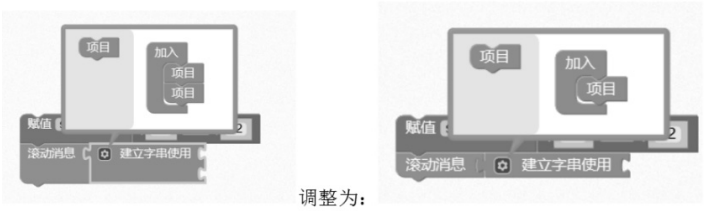


图 4-5　拼插“建立字符串使用”

图 4-6　选择变量“sumb”

⑤ 点击“下载 hex”按钮保存文件，然后将该文件拖到 TURNIPBIT 磁盘中，我们会看到 LED 灯的显示结果，看看得数是不是 35？

2．代码实现

我们还可以像程序员一样用纯代码来实现加法计算器。看下面的纯代码：

```
sumb=23+12    #定义与赋值 sumb
display.scroll("23+12=")  #显示"23+12="
display.scroll(str(sumb))  #显示 23 加 12 的得数 35
```

这个程序一共有 3 行代码，第 1 行是一个加法等式，程序会把右边的 23 加 12 的和赋值给左边的变量 sumb，同时声明 sumb 为整型变量。第 2 行和第 3 行主要用到 display.sccroll()和 str()，其中 display.scroll()前面讲过，滚动显示括号内的字符串；而 str()则是将括号内的数值转换为字符串。例如，str(123)相当于把整数 123 转换成"123"。

【思考】

用 TurnipBit 分别进行加、减、乘、除计算。

4.4　知识要点

4.4.1　拼插编程

【掌握】

- 变量的概念、命名以及类型。
- 变量的使用。

4.4.2　代码编程

【掌握】

转换字符串命令 str()。

第 5 章　会走的机器人

5.1　机器人是怎么走的

动画是通过把人物的表情、动作、变化等分解成许多动作瞬间的画幅，再用摄影机连续拍摄成一系列画面，在视觉上形成连续变化的图画。它应用的就是“视觉暂留”原理。“视觉暂留”是指人的眼睛看到一幅画或一个物体后，在 0.34 秒内不会消失。利用这一原理，在一幅画还没有消失前播放下一幅画，就会给人一种流畅的视觉变化效果。

想象一下交通信号灯中的绿灯小人是怎样行走的？在这里我们把它叫作机器人，那么根据“视觉暂留”这个原理它能分解成几个动画呢？

现在，我们就利用 TurnipBit 板载的 5×5 LED 点阵来让机器人走起来。

所需器材：

- TurnipBit 开发板一块
- 下载数据线一条
- 接入互联网的电脑一台（推荐使用 Google Chrome 或者 Firefox 浏览器）

5.2　循环转圈圈

在开始制作会走的机器人之前，我们先要学习一个知识，还记得第 3 章中的

“倒计时器”实验吗？先来看一下它的代码。

```
from microbit import *
display.show("9")
sleep(1000)
display.show("8")
sleep(1000)
display.show("7")
sleep(1000)
display.show("6")
sleep(1000)
display.show("5")
sleep(1000)
display.show("4")
sleep(1000)
display.show("3")
sleep(1000)
display.show("2")
sleep(1000)
display.show("1")
sleep(1000)
display.show("0")
sleep(1000)
```

从代码可以看出，display.show()和 sleep()形成了一个代码块，重复了 10 次，每次只是数字发生了有规律的变化。那么有没有更方便的方法呢？

看如下代码片段：

```
from microbit import *
for i in range(9,-1,-1):
    display.show(str(i))
    sleep(1000)
```

执行这段代码看看效果，这 4 行代码一样可以完成 21 行代码的工作。这里用到的知识就是循环。

5.2.1　for 循环

for 语句是重要的循环语句，其结构如下：

```
for 变量 in 序列:
```

```
    执行语句
```

（1）字符串序列，变量会从字符串中依次取出字符，然后进行循环。例如：

```
for i in "python":
    display.show(i)
```

这里 i 会从 python 中依次取出字符，LED 屏会显示 p、y、t、h、o、n。

（2）range(n,m,t)，数字的范围，表示从 n 到 m，间隔是 t 的所有数的范围，t 默认为 1。例如：

```
for i in range(1,9,2):
    display.show(str(i))
```

for 循环体内的语句执行了 5 次，LED 屏依次显示 1、3、5、7、9。

5.2.2 while 循环

（1）无限循环，在一些程序中，特别是对传感器等的操作中，经常会用到无限循环，也就是让程序一直不停地运行。例如：

```
while True:
display.show("H")
```

开发板会一直执行 display.show("H")这条语句。这里的 True 是逻辑关键词，表示“真”。如果用 False，则表示“假”，此时循环体内的语句永远不会被执行。

（2）当条件成立时循环。

```
while 条件:
    执行语句
```

当 while 后面的条件成立时，会执行 while 循环体内的语句。例如：

```
while i<10:
    display.show(str(i))
    i=i+1
```

当 i 小于 10 时，执行显示 i，然后 i+1，直到 i=10 循环停止。

5.2.3 continue 和 break

1．continue

continue 表示跳过当前的一次循环。比如从 0 循环到 10，每次循环执行显示

当前数字，当执行显示到 5 时，跳过 5，即不显示 5，直接进入显示 6 的循环中。代码如下：

```
i=0
while i<11:
    if i==5:
        continue
    else:
        display.show(str(i))
```

2．break

break 与 continue 不同，break 表示直接中断循环。比如在下面的代码中，即将上面代码中的 continue 换为 break，则只显示到数字 4，当 i 为 5 时，就跳出循环，后面的数字不再显示。

```
i=0
while i<11:
    if i==5:
        break
    else:
        display.show(str(i))
```

注意：在 Python 中，是通过缩进来区分代码块的。比如在 for 循环语句中，冒号后的块内语句全部向右缩进。代码缩进几个字符没关系，但同一个块内的语句一定要缩进相同。

5.3　画一个会走的机器人

5.3.1　使用“创建图像”拼画一个静止的机器人

① 在左侧的块选择区中找到“显示”块，用鼠标选中拼 

，拖曳至模块编辑区，如图 5-1 所示。

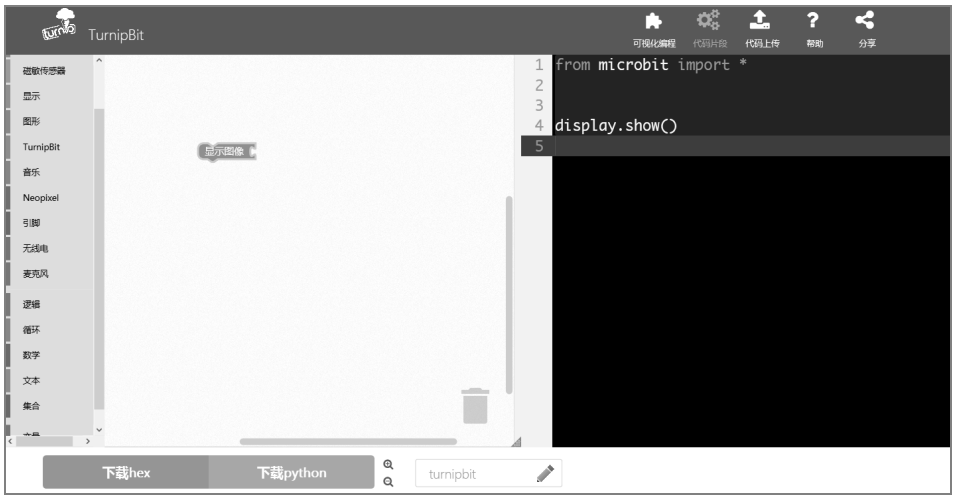


图 5-1　显示图像

② 在左侧的块选择区中找到“图形”块，用鼠标选中“创建图像”插  ，拖曳至模块编辑区，并与“显示图像”进行拼插，如图 5-2 所示。

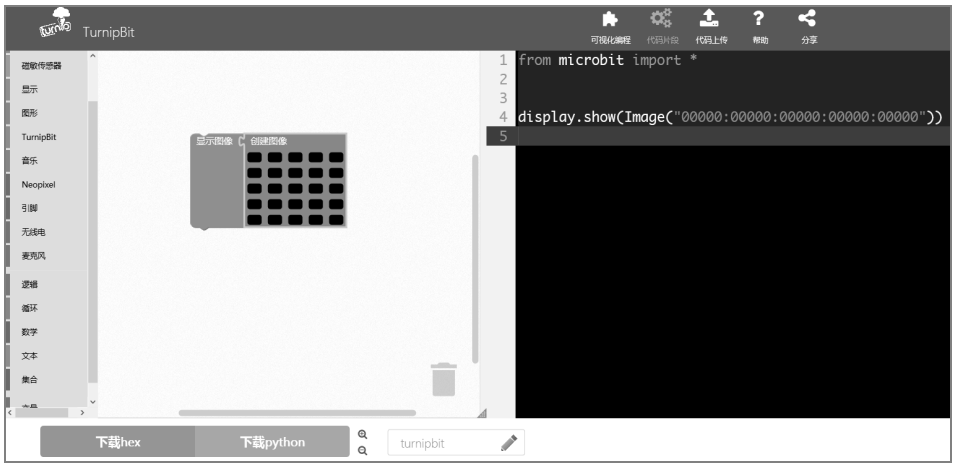


图 5-2　创建图像

③ 选择“创建图像”插的一个黑色区域，并选中蓝色，蓝色到黑色一共分 10 级，分别代表 LED 灯的亮度，这里我们选择最亮的蓝色。全部选择完毕后，效果如图 5-3 所示。

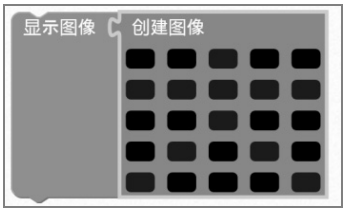


图 5-3　静止的机器人效果

④ 保持程序名 TurnipBit 不变，点击“下载 hex”按钮将程序保存到电脑中，如图 5-4 所示。

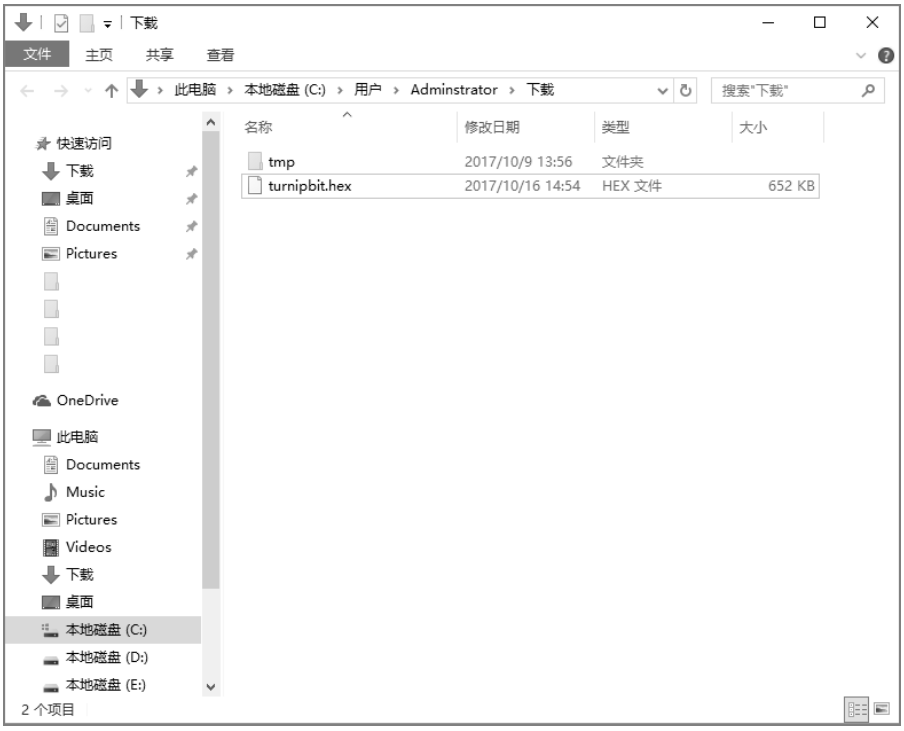


图 5-4　保存 HEX 文件

⑤ 把所保存的 HEX 文件拖入 TURNIPBIT 磁盘中，我们会看到 TurnipBit 板子上的灯在闪烁，说明正在下载到控制板中。下载成功后，我们会看到程序正常运行，效果如图 5-5 所示。

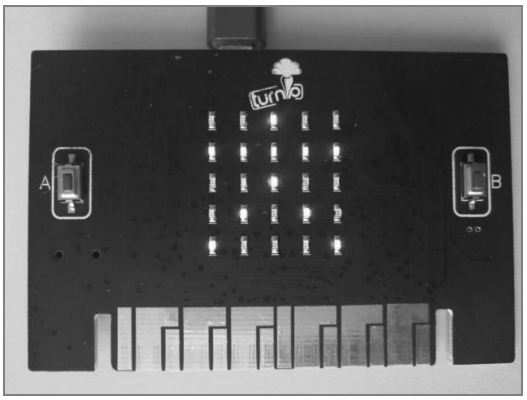


图 5-5　运行效果

【思考】

静止的机器人，怎样才能让它动起来呢？

5.3.2　使用“创建图像”让机器人动起来

如何才能让机器人动起来呢？

现在我们一起来分解一下机器人走路的动作。我们将机器人走路分为 3 个动作，即“立正”“左脚迈出”“右脚迈出”，如图 5-6 所示。将这 3 个动作连起来，并切换显示得快一些，在“视觉暂留”原理的作用下，绿灯机器人看似就动起来了。下面我们尝试通过编程来看看效果。

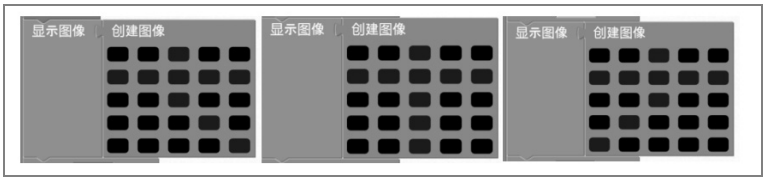


图 5-6　行走动作分解

① 在左侧的块选择区中找到“显示”块，用鼠标选中 显示图像 ，拖曳至模块编辑区。

② 在左侧的块选择区中找到“图形”块，用鼠标选中“创建图像”，插入“显示图像”中，并描画设定好的 3 个连接动作的机器人，如图 5-7 所示。

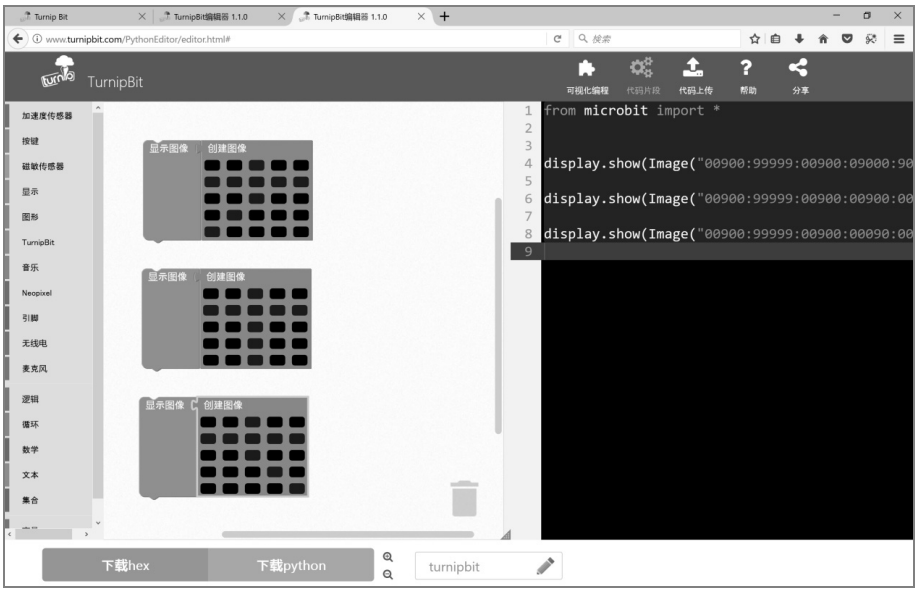


图 5-7　绘制行走动作

③ 在 3 个机器人后面加上一个“TurnipBit”中的“睡眠”，并把睡眠时间调整成 300 毫秒，如图 5-8 所示。

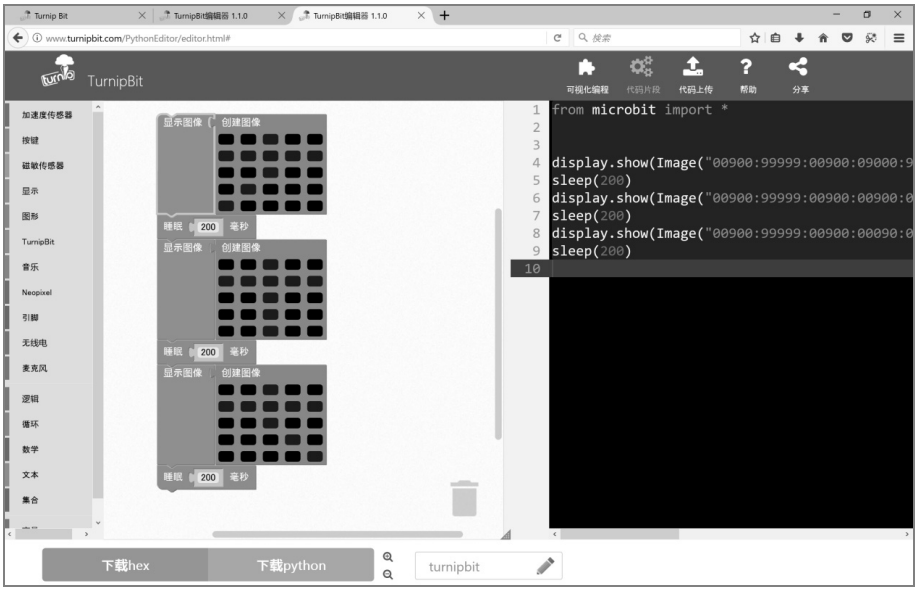


图 5-8　会动的机器人

④ 点击“下载 hex”按钮保存文件，并把所保存的 HEX 文件拖入 TURNIPBIT 磁盘中，我们会看到 TurnipBit 板子上的灯在闪烁，说明正在下载到控制板中。下载成功后，我们会看到程序正常运行，机器人也就走起来了。

【思考】

考虑加入无限循环，让机器人一直走下去。

5.3.3 让机器人一直走下去

在上面的例子中，机器人只走了一次，那么如何让机器人一直走下去呢？我们一起看看下面的例子。

如图 5-9 所示加入了两个新的拼插。

（1）“循环”中的“重复当”  ，这就是 while 循环。

（2）“逻辑”中的逻辑布尔值  ，表示无限循环。

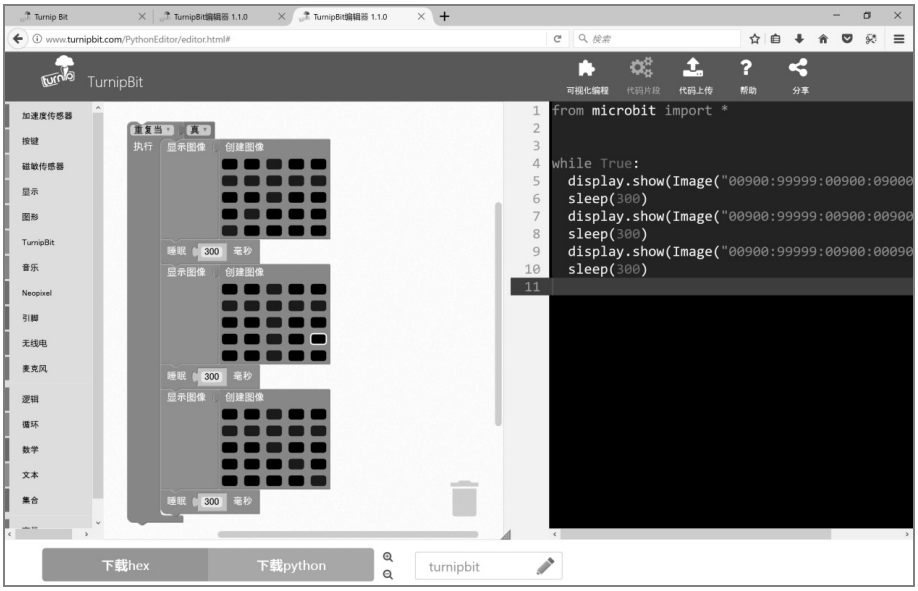


图 5-9 加入 while 循环和逻辑布尔值“真”

下面我们一起来分析程序。

while 循环开始后，先判断条件是否满足，如果满足就执行循环体内的语句，执行完毕后再回来判断条件是否满足，如此无限重复；直到条件不满足时，执行 while 循环后面的语句。因为条件是“真”，所以一直满足，机器人会一直走下去，如图 5-10 所示。

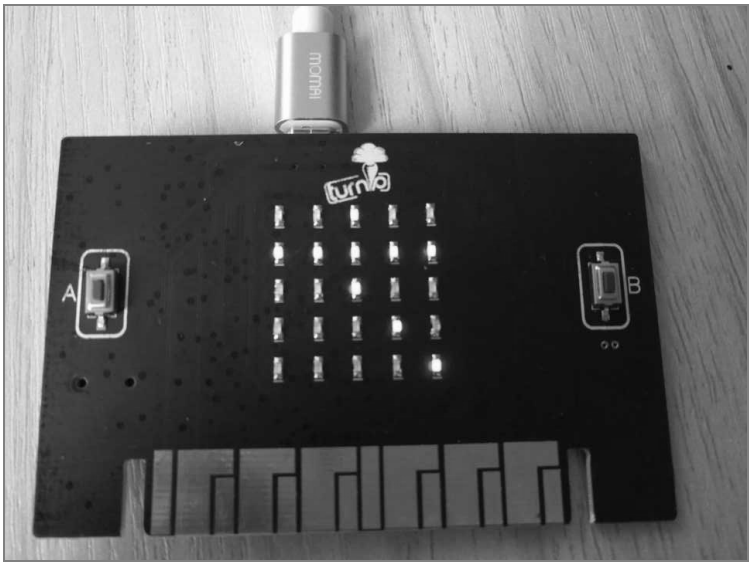


图 5-10　一直走的机器人运行效果图

那么该如何让机器人停下来呢？我们前面学到的 break 语句可以解决这个问题。

如图 5-11 所示，在右侧的这段程序中，当按键 A 按下时，LED 灯会滚动显示“Red Led”；松开后是绿灯，绿灯时机器人会走。机器人会走多长时间呢？“运行时间”running_time()函数决定了机器人的走动时间。这里设定 60 秒，通上电 60 秒内（running_time()<60000）机器人会一直走动，超过 60 秒后中断循环，机器人停止走动；在 60 秒内如果按键 A 处于按下状态，机器人会消失，会显示“Red Led”，即相当于红灯，如果松开按键 A，在 60 秒内机器人就会继续行走。试试这段代码，看看你还能做哪些改进？

在这里介绍一下逻辑拼插。逻辑块中的“如果”就是分支判断结构，如果满足后面的条件，就执行“执行”内的语句（关于条件语句，具体见第 6 章）。在本

例中，“按键 A 被按下”的条件，就是当按键 A 被按下时触发执行语句的意思。当按键 A 被按下后，滚动显示消息“Red Led”。

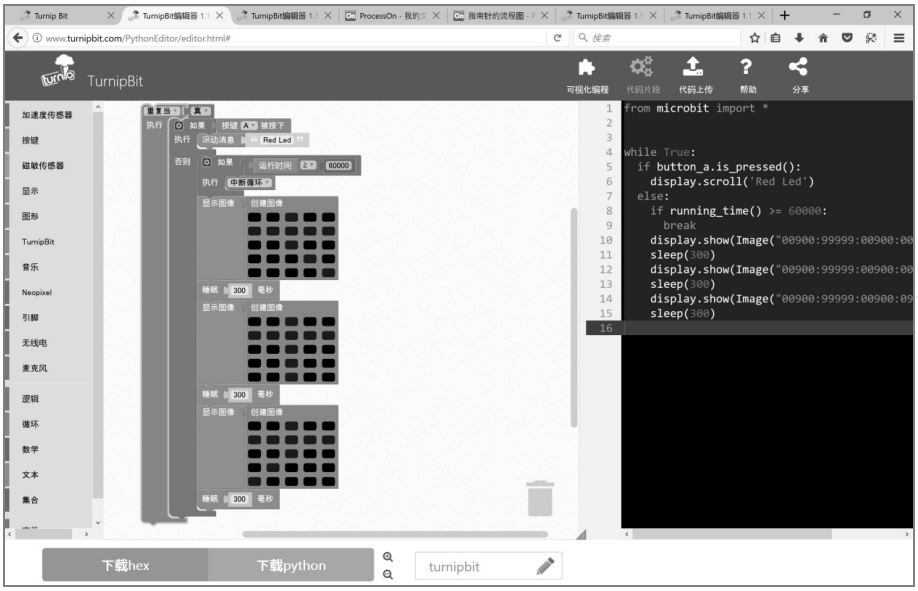


图 5-11　运行 60 秒后中断循环

TurnipBit 有两个按键，即按键 A 和按键 B，每个按键有三种状态。

- “按键 A 被按下”：按键 A 被按下的状态。
- “按键 A 曾经按下”：按键 A 曾经按下过，与现在的状态无关。
- “按键 A 曾经按下过的次数”：返回的是数字，表示按键 A 曾经按下过的次数，当达到这个条件时就会执行指定的语句。

5.3.4　画出会走的机器人的流程图

下面给出伪代码。

```
Begin（算法开始）
循环体开始
    如果按键 A 被按下
        显示“Red Led”（现在是红灯）
    否则
```

```
        如果执行时间大于 60 秒
            结束循环，跳出循环体
        右腿在后面
        睡眠 0.3 秒
        立正
        睡眠 0.3 秒
        左腿在前面
        睡眠 0.3 秒
循环体结束
End （算法结束）
```

根据伪代码画出流程图，如图 5-12 所示。

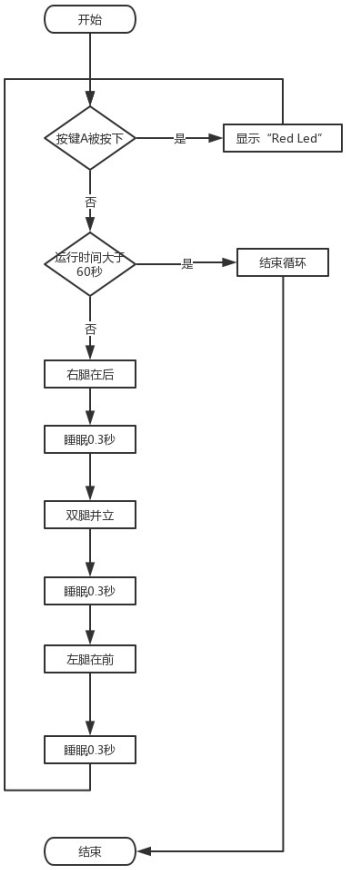


图 5-12　会走的机器人流程图

【思考】

如何让机器人走 30 秒再停止呢？

5.4 知识要点

5.4.1 拼插编程

【掌握】

- 中断的概念。
- 会使用“睡眠”。
- 会使用“中断循环”。

【理解】

- 动画原理。
- for、while 循环的应用。
- continue、break 的使用。

5.4.2 代码编程

【掌握】

- 循环的用法。
- 中断的用法。

【理解】

- 睡眠的用法。
- 运行时间的用法。
- 按键的状态。

第 6 章　好玩的掷骰子游戏

6.1　掷骰子游戏

骰子，中国古代民间娱乐用来投掷的博具。骰子通常作为桌上游戏的小道具，最常见的是六面骰，它是一个正立方体，上面分别有 1~6 个孔（或数字），其相对两面的数字之和必为 7。中国的骰子习惯在一点和四点漆上红色。骰子是容易制作和取得的乱数产生器。

现在，我们就利用 TurnipBit 板载的 5×5 LED 点阵来完成这样一个古老的小游戏。

所需器材：

- TurnipBit 开发板一块
- 下载数据线一条
- 接入互联网的电脑一台（推荐使用 Google Chrome 或者 Firefox 浏览器）

6.2　学会做选择题

在上一章中，我们在做会走的机器人的时候，用到了“如果运行时间超过 60 秒”。这个“如果”就是在做选择题。在编程过程中，我们经常会进行选择决策。下面给出几个例子。

如果 10 道题全做对，那么就得 100 分。

如果击中对方飞机，对方飞机就消失，同时得分加 1 分。

如果到了黑夜，那么小夜灯亮起。

程序在执行时，就是对“如果”条件进行检测。如果为“真”，那么做出相应的决策；如果为“假”，那么做出其他决策。

6.2.1 逻辑运算

任何程序设计语言都具有逻辑运算这个基本功能，Python 也一样。说到逻辑运算，首先要学会判断“真”和“假”。例如，10 小于 20，就是“真”，而相反就是“假”。逻辑运算的结果返回均为“真”或者“假”，“真”用 True 表示，“假”用 False 表示。在 TurnipBit 编程界面的“块”中有“逻辑”模块，这里包含了当前拼插编程支持的逻辑运算符。

1. 比较运算符

比较运算符在“逻辑”模块内，形式如图 6-1 所示。

图 6-1 比较运算符

假设变量 a 为 5，b 为 10，比较运算符的含义如表 6-1 所示。

表 6-1 比较运算符的含义

运算符	描　述	实　例
==	等于，比较对象是否相等	(a == b)返回 False
!=	不等于，比较两个对象是否不相等	(a != b)返回 True
<>	不等于，比较两个对象是否不相等	(a <> b)返回 True 这个运算符类似于!=
>	大于	(a > b)返回 False
<	小于	(a < b)返回 True
>=	大于或等于	(a >= b)返回 False
<=	小于或等于	(a <= b)返回 True

2．逻辑运算符

逻辑运算符也在“逻辑”模块内，形式如图 6-2 所示。

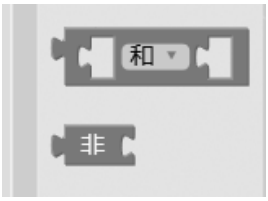


图 6-2　逻辑运算符

假设变量 a 为 1，b 为 0，逻辑运算符的含义如表 6-2 所示。

表 6-2　逻辑运算符的含义

运算符	逻辑表达式	描　　述	实　　例
and	x and y	布尔“与”，如果 x 为“假”，则返回“假”；否则，返回 y 的值	(a and b)返回 0
or	x or y	布尔“或”，如果 x 是“真”，则返回 x 的值；否则，返回 y 的值	(a or b)返回 1
not	not x	布尔“非”，如果 x 为“真”，则返回“假”；如果 x 为“假”，则返回“真”	not(a and b)返回 1

6.2.2　if 判断语句

在 Python 编程中，if 语句用于控制程序的执行。其基本形式为：

```
if 判断条件 1:
   执行语句 1
elif 判断条件 2:
   执行语句 2
elif 判断条件 3:
   执行语句 3
else:
   执行语句 4
```

对应到拼插编程中，就是如图 6-3 所示的这个模块。在这个模块中，你可以任意拖入不同的配置，对应的程序分支有以下几种。

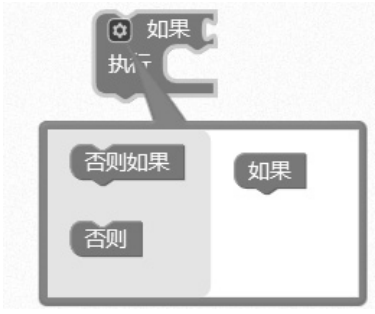


图 6-3　判断语句中的拼插编程

1.“如果，那么”

这是判断语句中最简单的方式，如图 6-4 所示。如果条件满足，那么执行“如果”条件块内的语句。对应的 if 语句格式为：

```
if (条件):
    pass
```

图 6-4　判断语句“如果，那么”

2.“如果，那么，否则”

判断语句“如果，那么，否则”，需要在“如果”配置中加入“否则”，如图 6-5 所示。如果条件满足，就执行“那么”块内的语句；如果条件不满足，那么执行“否则”块内的语句。对应的 if 语句格式为：

```
if (条件):
    pass
else:
    pass
```

图 6-5　判断语句“如果，那么，否则”

3.“如果，那么，否则如果，那么，否则”

判断语句“如果，那么，否则如果，那么，否则”，如果“条件 1”满足，就执行“那么”语句；如果“条件 1”不满足，但是满足“否则如果”“条件 2”，就执行“否则如果”下面的“那么”语句；如果上面的条件都不满足，那么执行“否则”后的语句，如图 6-5 所示。对应的 if 语句格式为：

```
if (条件 1):
    pass
elif(条件 2):
    pass
else:
    pass
```

图 6-6　判断语句“如果，那么，否则如果，那么，否则”

6.3　实现掷骰子游戏

6.3.1 绘制流程图

在掷骰子游戏中，如果按下 A 键，则表示开始掷骰子；如果按下 B 键，则表示显示骰子的点数。与前面几个实验一样，我们先写出这个游戏程序的伪代码。

```
Begin（算法开始）
变量 num （用于存储随机得到的骰子的点数）
变量 flag （作为按键标识位使用，1 为按下 A 键，开始掷骰子；0 为按下 B 键，显示掷骰子得到的点数）
无限循环（while）
    判断变量 flag 是否为 1
        如果按下按键 A，则将随机生成的 1~6 点数赋值给变量 num
    显示变量 num 至 LED 屏
    按键 A 按下
        将变量 flag 赋值为 1
    按键 B 按下
        将变量 flag 赋值为 0
End （算法结束）
```

根据伪代码画出流程图，如图 6-7 所示。

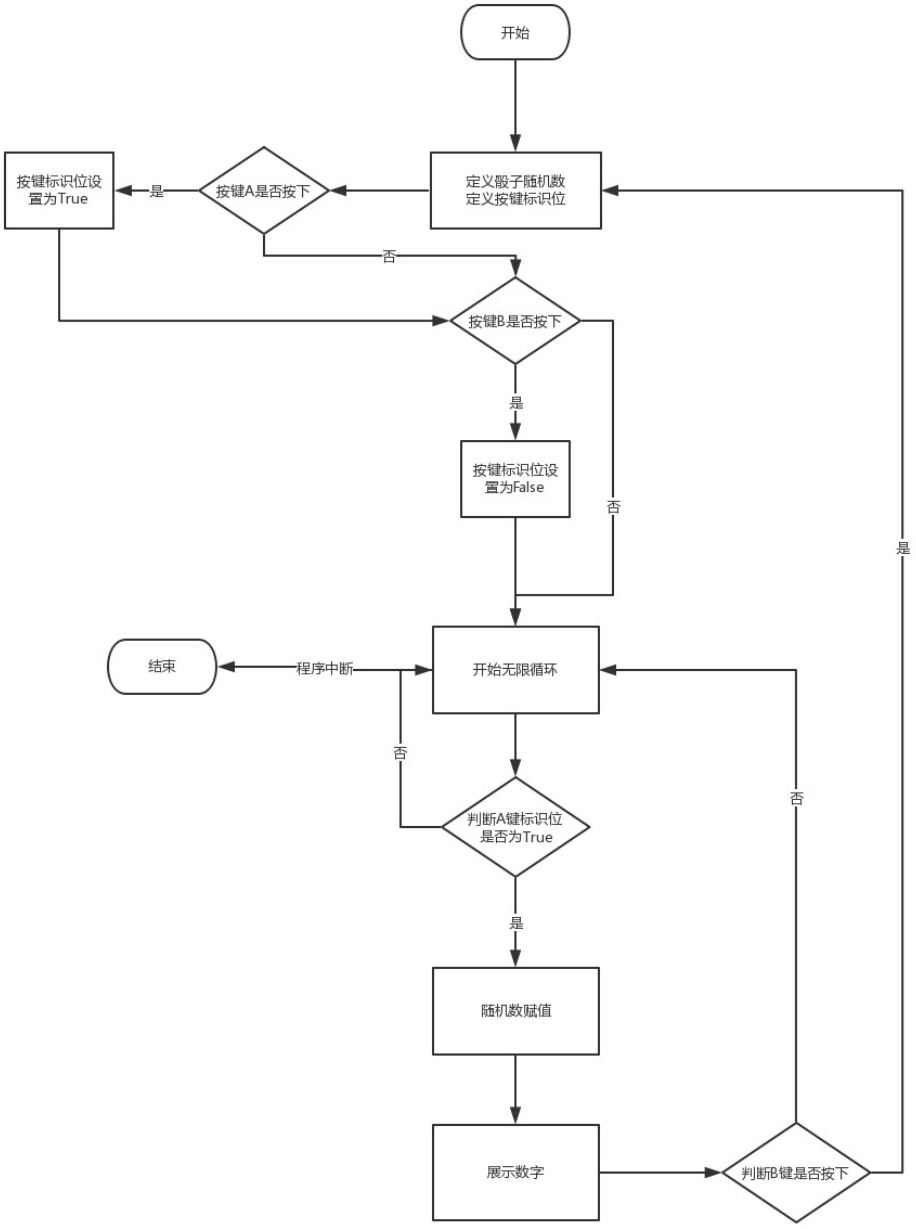


图 6-7　掷骰子游戏流程图

6.3.2　拼插编程实现掷骰子游戏

① 打开官方网站 http://www.turnipbit.com/，点击“开始编程”按钮进入编程界面，进行基础的模块化拼插编程的学习。

② 创建一个变量用来存储所显示的数字。选择“变量”块中的“创建变量”，点击“输入变量名”，这里设为 num，如图 6-8 所示。

图 6-8　创建变量

③ 给 num 设置一个初始值 1（num=1）。选择“赋值 num 到”（默认值为 0），然后选择“数学”块中的 0 和“赋值 num 到”拼插起来，修改数值为 1，如图 6-9 所示。

图 6-9　设置 num 为 1

④ 再创建一个新变量 flag，并赋值为 1，用来标识按键，如图 6-10 所示。

图 6-10　设置 flag 为 1

⑤ 放入一个循环，条件永远为“真”，让程序一直运行。选择“循环”块中的“重复当执行”，如图 6-11 所示。然后，选择“逻辑”块中的“真”，与循环拼插对接起来，如图 6-12 所示。

图 6-11　选择“重复当执行”

图 6-12　插入条件“真”

⑥ 在循环体内添加执行的内容。首先逻辑判断 flag 的值，选择“逻辑”块中的“如果执行”放到循环体内，如图 6-13 所示。

⑦ 在“逻辑”块中选择“等号”，设置判断条件“flag==1”，如图 6-14 所示。

图 6-13　加入判断语句

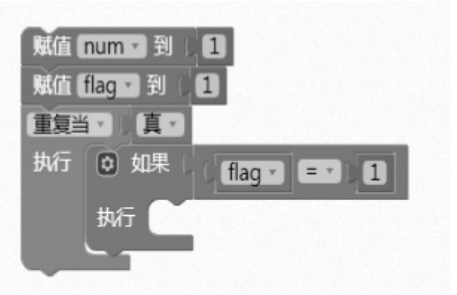


图 6-14　设置判断条件“flag==1”

⑧ 当 flag 为 1 时，给 num 赋值 1~6 随机数并显示。选择“变量”块中的“赋值 num 到”，再选择“数学”块中的“从 1 到 100 之间的随机整数”，与“赋值 num 到”拼接，并修改为从 1 到 6 之间的随机整数，如图 6-15 所示。

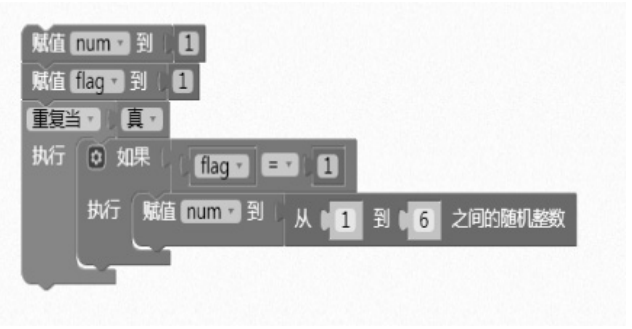


图 6-15　生成随机数

⑨ 显示 num 的值。在显示之前，先将已有的显示内容清除，选择“显示”块的“清除显示内容”放到逻辑执行体的下面与其对接；选择“显示”块的“显示图像”放到“清除显示内容”下面；选择变量“num”与“显示图像”拼插在一起，如图 6-16 所示。在这里我们要注意，num 是整数型，而前面讲过“显示图像”（display.show()）只能显示字符串，所以要使用 str()函数（见图 6-19）将 num 转换为字符串型，这个转换只能在代码显示区进行修改，将 display.show(num)修

改为 display.show(str(num))。

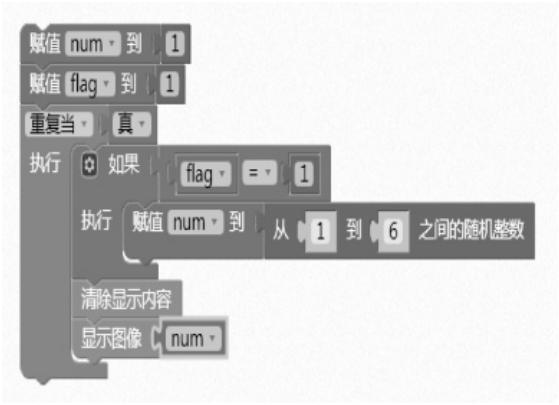


图 6-16　显示 num 的值

⑩ 判断按键 A 是否被按下，在“显示图像”的下方添加“如果 执行 ”。在“如果”条件中处添加“按键”块中的“按键 A 被按下”，在“执行”中为 flag 赋值 1，表示重新生成随机数，如图 6-17 所示。

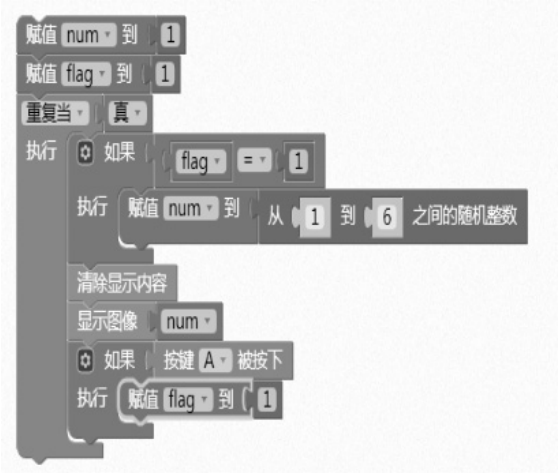


图 6-17　按下按键 A，重新生成随机数

⑪ 继续添加“如果 执行 ”，在“如果”条件中添加“按键 B 被按下”，在“执行”中为 flag 赋值，如图 6-18 所示。

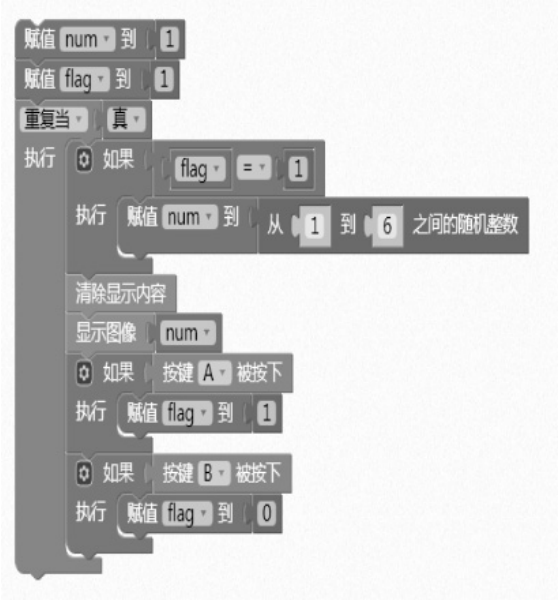


图 6-18　按下按键 B，显示结果

⑫ 到上一步，其实程序就已经完整了，但这时会发现运行时数字变动太快，甚至有些像乱码。这是什么原因导致的呢？其实就是显示图像这里停顿的时间太短，我们在“显示图像”下面加入“睡眠 1000 毫秒”，把 1 000 毫秒修改为 150 毫秒，于是完整的代码如图 6-19 所示。

```
1  import random
2  from microbit import *
3
4
5  num = 1
6  flag = 1
7  while True:
8    if flag == 1:
9      num = random.randint(1, 6)
10   display.clear()
11   display.show(str(num))
12   sleep(150)
13   if button_a.is_pressed():
14     flag = 1
15   if button_b.is_pressed():
16     flag = 0
```

图 6-19　掷骰子游戏的完整代码

注意图中框内部分，因为 num 为整数型，所以需要先将 num 转换为字符串型。

⑬ 将 HEX 文件下载到 Turnipbit 中，让我们一起掷骰子吧。

【思考】

为什么 display.show 这个函数不能直接显示 num?

6.4 代码分析

6.4.1 基本原理

随机产生 1~6 整数，用来模拟骰子的 6 个面。数字不断地滚动，用来表示骰子的翻滚，通过按键来停止或开始翻滚，形成掷骰子、停止、再次掷骰子的效果。

6.4.2 逻辑分析

```
#代码
import random
from microbit import *
rtum = 1
fiag = 1
while True:
    if flag == 1:
        rum = random. randint(l, 6)
    display.clear()
    display.show(str(num))
    sleep(150)
    if button_a.is_pressed():
        fiag = 1
    if button_b.is_pressed():
        fiag = 0
```

程序入口为 while 循环语句，首先进行“if 逻辑判断”，判断变量 flag 是否为 1。若变量 flag 为 1，则生成一个 1~6 的随机数并赋值给 num。然后清空当前的显示，将 num 转换为字符串并展示，延时 150 秒。接下来再进行“if 逻辑判断”，如果按键 A 被按下，则 flag 的值改为 1。对按键 B 进行判断，如果按键 B 被按下，

则将 flag 的值改为 0。由此可得出结论：flag 为按键的标识位，当按键 A 被按下时，骰子进行翻滚；当按键 B 被按下时，骰子停止翻滚，即显示当前的点数。在程序中添加了清空当前屏幕显示的代码，使得程序的可读性更好。

6.5　知识要点

6.5.1　拼插编程

【掌握】

- TurnipBit 拼插编程网站的代码切换。
- 变量的创建及赋值。
- while 及判断的使用。

6.5.2　代码编程

【掌握】

if 判断语句的多种形式。

第 7 章　无线投票器

7.1　制作无线投票器

昨天，班长就某项集体活动征求大家意见，采用的方法还是传统的做法，同意的举手，然后数数同意的同学数量。今天，老师在操场上征求全校 2000 名学生的意见，如果也采用举手的方法，是不是要数很长时间呢？于是，投票器就发挥了作用，每个同学手上都有一个投票器，同意按 A，不同意按 B，然后大屏幕上马上就能显示出同意的有多少人，不同意的有多少人，听起来这个创意是不是很酷？

现在，我们就利用 TurnipBit 来制作一个无线投票器，让它来帮助完成投票统计的工作。在这个实验中，我们将用到 TurnipBit 上的无线模块，需要两块 TurnipBit 同时工作，一块用来做统计终端，一块用来模拟投票器。

所需器材：

- TurnipBit 开发板两块
- 下载数据线一条
- 接入互联网的电脑一台（推荐使用 Google Chrome 或者 Firefox 浏览器）

7.2　准备知识

7.2.1　函数

1．函数的定义

在前面章节中，如果仔细看程序，就会发现程序里面有些代码或者代码块被重复多次使用，比如在第 3 章“倒计时”中，数字每减少 1，就停 1 秒，这两个动作就是重复性的，我们就可以把这个重复性的代码块定义为一个函数，在编程时，每次只需调用这个函数即可。比如经常使用的 display.show()就是 display 对象的一个函数，在编程时不需要知道 display.show()这个函数的源代码是如何实现的，只需要知道如何调用它就可以了。

Python 使用 def 来定义函数，语法格式如下：

```
def 函数名( 参数 ):
    代码块
    return [expression]
```

在函数的定义过程中，具体规则如下：

- 函数块以关键字 def 开头，后跟函数名和小括号（()）。
- 函数分为带参数和不带参数的两种，对于带参数的函数，其参数应放置在小括号中，也可以在小括号中定义参数。
- 每个函数中的代码均以冒号（:）开始，并缩进。
- 语句 return [expression]用于退出一个函数，可选地将一个表达式传回给调用者。如果没有使用参数的 return 语句，则它与 return None 相同。

2．函数的使用

这里我们重点学习不带参数和带参数的函数，以及无返回值的有返回值的函数的使用。

（1）不带参数的函数

这类函数用来完成某项工作，无须返回值。比如定义一个函数显示 k，在程

序中只要显示 k，就调用这个函数。代码如下：

```
from microbit import *

def display_k():
    display.show("k")
sleep(1000)
display.show("9")
sleep(1000)
display_k()  #调用函数
display.show("0")
sleep(1000)
```

上面定义了一个函数 display_k()，该函数的作用是显示 k，并停 1 秒。在这段程序中，显示 9 以后，调用了 display_k()函数，于是该程序的运行效果是 TurnipBit 先显示 9，停 1 秒，再显示 k，停 1 秒，最后显示 0。

（2）带参数的函数

还是使用上面的这段程序，如果要将 display_k()函数修改为一个可以显示任意字符的函数，应该怎么做呢？这时就需要用到带参数的函数，程序在运行时，将需要显示的字符通过参数传递给函数，从而达到目的。

```
from microbit import *
def display_k(str_a):
    display.show(str_a)
    sleep(1000)

nstr="h"   #定义 nstr 为字符串型，并赋初始值"h"
display.show("9")
sleep(1000)
display_k(nstr)  #调用函数，将 nstr 传给 str_a，从而显示“h”
nstr="r"  #为 nstr 赋值"r"
display_k(nstr)  #调用函数，显示“r”
display.show("0")
```

这段程序的运行效果是，首先 9 显示 1 秒，然后 h 显示 1 秒，r 显示 1 秒，最后显示 0。为什么 nstr 与 str_a 参数的名称不一致也能相互传递呢？在定义函数的时候，“def 函数名(参数)”中的参数称为形式参数，简称“形参”，用于接收调用该函数时传入的参数。在本程序的函数定义中，str_a 就是形参。在调用函数时，

使用的参数叫实际参数，又称“实参”。本程序中的 nstr 就是实参。实参与形参可以名称不一样，但是类型必须相同，即本程序中 str_a 是字符串，那么在定义实参 nstr 时也要用字符串型，否则程序就会出错。

（3）有/无返回值的函数

根据有无返回值，函数又可分为有返回值的函数和无返回值的函数，它们的明显区别在于函数体代码块后面的 return，如果 return 返回了实际的值，则为有返回值的函数；否则为无返回值的函数。在无返回值的情况下，return 可以省略。上面程序中的函数均为无返回值的函数，省略了 return。display_k()函数只是用于完成显示任务，而不需要返回相应的值。下面举一个需要返回值的例子，以第 4 章“方便的加法计算器”为例，定义 tsum 函数来计算加法，主程序用来显示结果。代码如下：

```
from microbit import *

def  tsum(numb1,numb2):
    sumb=numb1+numb2
    return sumb
fnum=23   #定义加数
bnum=12     #定义被加数
display.scroll("23+12=")     #滚动显示“23+12=”
hsum=tsum(fnum,bnum)  #调用 tsum()函数计算 fnum 与 bnum 的和，然后将结果返回给 hsum
display.scroll(str(hsum))     #滚动显示 hsum ，即求和的结果
```

在程序中，hsum=tsum(fnum,bnum)实现了对 tsum 的调用，同时完成了两个数的加法计算，并将加和的结果返回给 hsum。从程序可以看出，有返回值的函数在完成工作任务后，会将结果返回给主程序。

7.2.2　TurnipBit 无线模块的使用

就像手机、蓝牙音响等设备一样，TurnipBit 本身也带有无线模块，该模块可以实现多块 TurnipBit 板子之间的相互连接与通信，分为接收部分与发送部分。例如有两块 TurnipBit，一块负责发送消息，一块负责接收消息。其使用步骤如下：

① 打开“块”→“无线电”→“打开蓝牙”。

② 配置蓝牙参数，选择“无线电”→“配置蓝牙消息…”，在这里可以进行多块 TurnipBit 板子的配对。在配置各参数时，多块板子配置一样就可以了。

③ 配置 TurnipBit 是发送数据还是接收数据，发送就用“发送消息”，接收就用“接收消息”。

④ 关闭通信，选择“关闭蓝牙”。

例如：定义一块 TurnipBit 为发送机，每隔 1 秒发送一次“hello”。再定义一块 TurnipBit 为接收机，接收到“hello”后，滚动显示出来。

1．发送机

（1）拼插编程

发送机拼插编程如图 7-1 所示。

图 7-1　发送机拼插编程

（2）代码编程

具体解释见代码中的注释部分。

```
   import radio  #调用 radio 包
   from microbit import *
   radio.on()   #打开蓝牙
   radio.config(length=32,queue=3,channel=7,power=5,data_rate=radio
.RATE_1MBIT)  #配置蓝牙参数
   while True:          #每秒发送一次“hello”
        radio.send('hello')
```

```
        sleep(1000)
    radio.off()    #关闭蓝牙
```

2．接收机

（1）拼插编程

接收机拼插编程如图 7-2 所示。

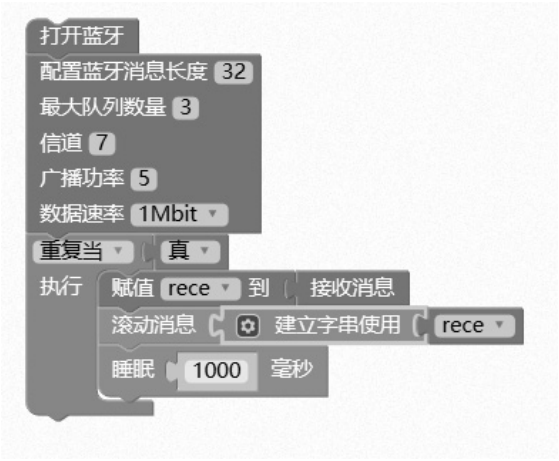


图 7-2　接收机拼插编程

（2）代码编程

代码解释见注释部分。

```
    import radio
    from microbit import *
    radio.on()    #打开蓝牙
    radio.config(length=32,queue=3,channel=7,power=5,data_rate=radio
.RATE_1MBIT)  #配置蓝牙参数
    while True:
        rece = radio.receive()     #接收数据
        display.scroll((str(rece)))    #滚动显示
        sleep(1000)
    radio.off()
```

上面两段代码的开头都有一行“import radio”，意思是调用 radio 包。另外，本书中每段代码都有一行“from microbit import *”。讲到这里，我们需要简单了解一下 Python 编程中“包”的概念。在 Python 中，为了编程方便，往往会设计

或者内嵌多个库，形成库文件，每个库可以理解为一类任务的集合。比如 radio 库就是用于控制无线传输的； microbit 库用于控制 TurnipBit 的显示、输出、输入等。在使用库时，在程序开头要先导入库。导入库的方法有两种：

```
import 库名
from 库名 import *
```

如果用第一种导入方法，当在编程过程中需要调用库里的方法或者函数时，还需要加上库名。例如：

```
import radio
radio.on()
```

这里调用了 radio 库里的 on()，需要用 radio.on()的形式。而对于“from 库名 import *”导入方法，调用时则可以省略库名。例如：

```
form microbit import *
display.show()
```

对于“from microbit import *”，那么调用 display 时可以直接使用 display.scroll()；而对于“import microbit”，那么调用 display 时需要使用 microbit.display.scroll()。

7.3 动手制作无线投票器

7.3.1 无线投票器流程图设计

这里我们用两块 TurnipBit 来实现无线投票器，其中 TurnipBit1 作为接收机，用来进行投票统计，当接收到“同意”票时，同意人数加 1，当接收到“反对”票时，不同意人数加 1；TurnipBit2 作为发送机，用来进行投票，按下 A 键表示“同意”，按下 B 键表示“反对”。下面用伪代码来分析发送机和接收机的程序设计。

1. 发送机

当按下 A 键时，发送“Y”表示“同意”；当按下 B 键时，发送“N”表示“反对”。程序的实现步骤为：

① 打开蓝牙。

② 配置蓝牙参数，蓝牙消息长度设为 32，最大队列数量设为 3，信道设为 7，广播功率设为 5，数据速率设为 1Mbit。

③ 进入无限循环中，显示内置图像“高兴”，表示准备好了，等待发送投票指令按下 A 键，发送“Y”，表示“同意”；按下 B 键，发送“N”，表示“不同意”。

具体流程图如图 7-3 所示。

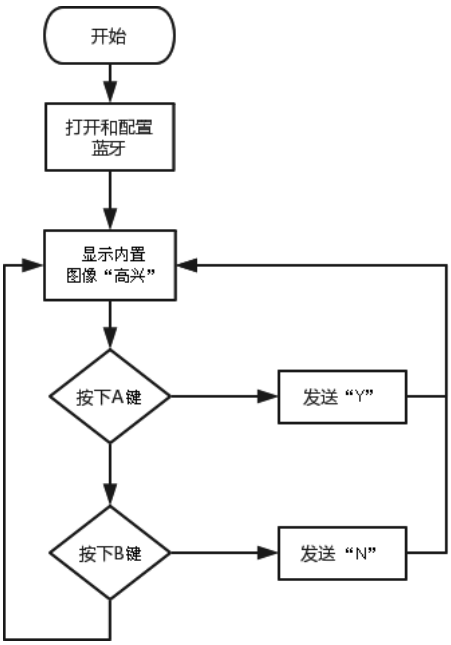


图 7-3 “投票器”发送机流程图

2．接收机

当接收机收到“Y”时，“同意”人数增加 1；当收到“N”时，“反对”人数增加 1，最后显示统计结果。程序的实现步骤为：

① 打开蓝牙。

② 配置蓝牙参数，蓝牙消息长度设为 32，最大队列数量设为 3，信道设为 7，广播功率设为 5，数据速率设为 1Mbit。

③ 准备好后，显示内置图像“是”，表示准备好了，等待接收发送机发来的

数据。

④ 按下 A 键，显示当前统计结果。

⑤ 如果收到“Y”，“同意”人数加 1，显示当前统计结果；如果收到“N”，“反对”人数加 1，显示当前统计结果。

具体流程图如图 7-4 所示。

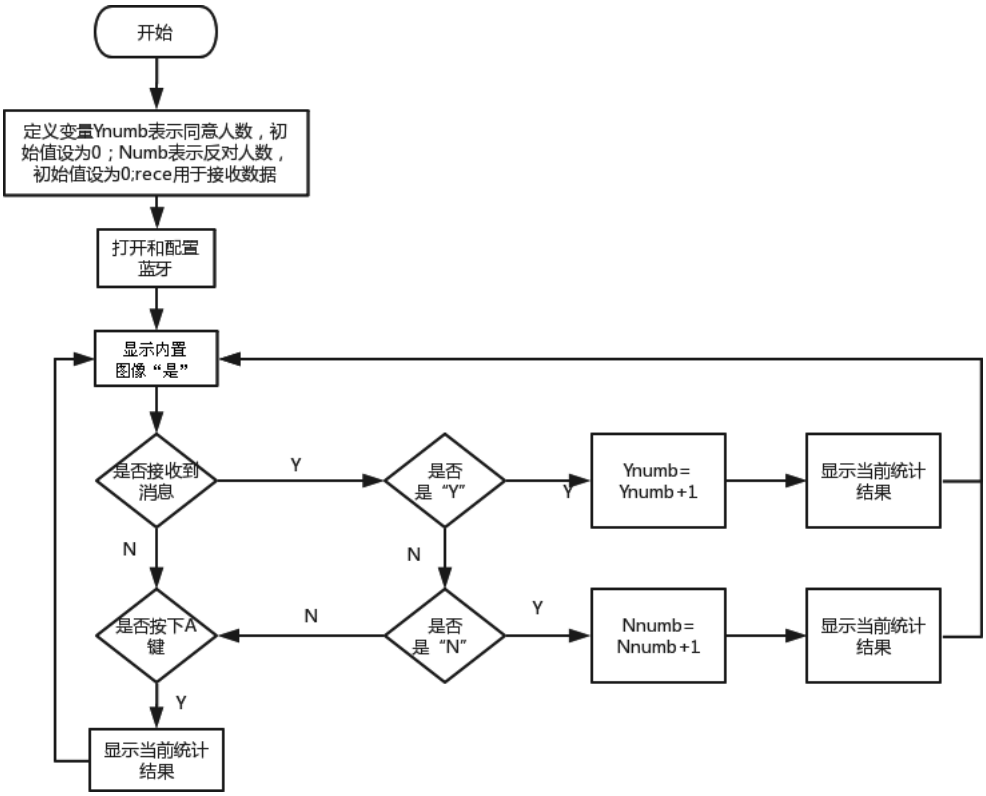


图 7-4 “投票器”接收机流程图

7.3.2 无线投票器程序实现

1. 发送机程序实现

（1）拼插编程

根据图 7-3 所示的流程图，发送机的拼插编程主要步骤如下：

① 在块选择区中选择“无线电”模块，点击“打开蓝牙”，如图 7-5 所示。

图 7-5　打开“蓝牙”

② 选择“配置蓝牙消息…”，修改“广播功率”为 5，如图 7-6 所示。

③ 选择“重复当”循环，从“逻辑”块中选择“真”，形成无限循环，如图 7-7 所示。

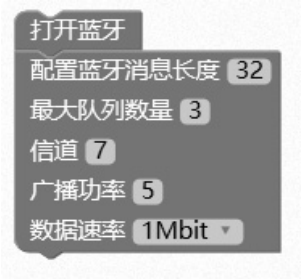


图 7-6　修改配置

图 7-7　加入循环

④ 选择“逻辑”块中的“如果　执行　”，将其配置为“如果 执行 否则如果 执行 ”，分别在第一个条件中加入“按键 A 被按下”，在第二个条件中加入“按键 B 被按下”，如图 7-8 所示。

图 7-8　判断按键 A 或者按键 B 是否被按下

⑤ 在“按键 A 被按下”条件中执行“滚动消息‘Y’”和“无线电”块中的“发送消息‘Y’”，在“按键 B 被按下”条件中执行“滚动消息‘N’”和“无线电”块中的“发送消息‘N’”，如图 7-9 所示。

图 7-9　发送字符

⑥ 在循环体内最后加入显示“内置图像‘高兴’”，形成完整的程序，如图 7-10 所示。

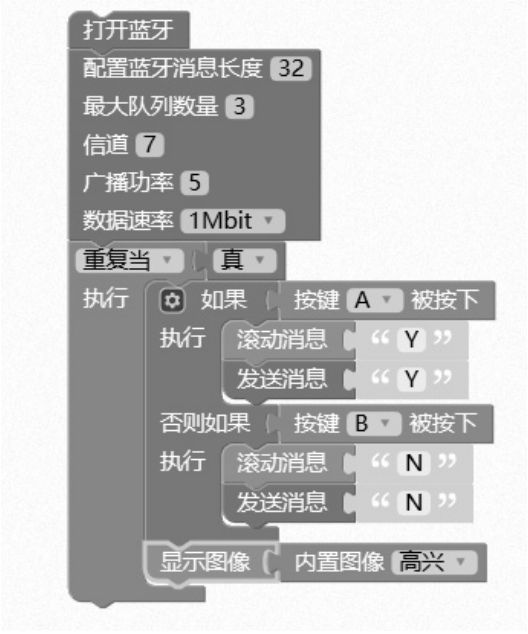


图 7-10　显示图像

（2）代码编程

通过上面拼插编程形成的全部代码如下：

```
    import radio
    from microbit import *
    radio.on()   #打开蓝牙
    radio.config(length=32,queue=3,channel=7,power=5,data_rate=radio
.RATE_1MBIT)  #配置蓝牙参数
    while True:
        if button_a.is_pressed():  #如果按键 A 被按下
              display.scroll('Y')
              radio.send('Y')     #发送“Y”
        elif button_b.is_pressed():   #如果按键 B 被按下
              display.scroll('N')
              radio.send('N')    #发送“N”
        display.show(Image.HAPPY)  #显示内置图像“高兴”
```

从这段代码中我们不难发现，当按键 A 或者按键 B 被按下时，执行的语句是相同的，只是参数不同。那么是否可以用函数来实现，以使代码更加简洁呢？转换为函数的代码如下：

```
import radio
from microbit import *
def sendstr(nstr): #控制发送的函数
    display.scroll(nstr)
    radio.send(nstr)
radio.on()  #打开蓝牙参数
radio.config(length=32,queue=3,channel=7,power=5,data_rate=radio
.RATE_1MBIT)  #配置蓝牙参数
while True:
    if button_a.is_pressed():  #如果按键 A 被按下
        sendstr('Y')
    elif button_b.is_pressed():  #如果按键 B 被按下
        sendstr('N')
    display.show(Image.HAPPY)  #显示内置图像“高兴”
```

2．接收机程序实现

（1）拼插编程

根据图 7-4 所示的流程图，接收机的拼插编程主要步骤如下：

① 首先定义 Ynumb、Nnumb 和 rece 三个变量，并给 Ynumb 和 Nnumb 赋初始值 0，如图 7-11 所示。具体变量定义的方法可参见第 4 章。

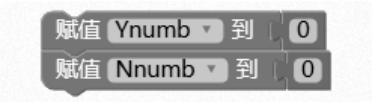


图 7-11　定义变量

② 选择“打开蓝牙”和“配置蓝牙消息…”，这里同样要修改蓝牙的“广播功率”为 5，如图 7-12 所示。

图 7-12　修改“蓝牙”配置

③ 建立无限循环，在循环内接收消息并赋值给 rece。然后对 rece 进行分析，如果 rece 是'Y'，那么 Ynumb 加 1，同时“滚动显示”Y 的值和 N 的值；如果 rece 是'N'，那么 Nnumb 加 1，同时“滚动显示”Y 的值和 N 的值。如果按键 A 被按下，则查看当前统计状态，“滚动显示”Y 的值和 N 的值。如果没收到消息，也没按下按键 A，则正常显示内置图像“是”，如图 7-13 所示。

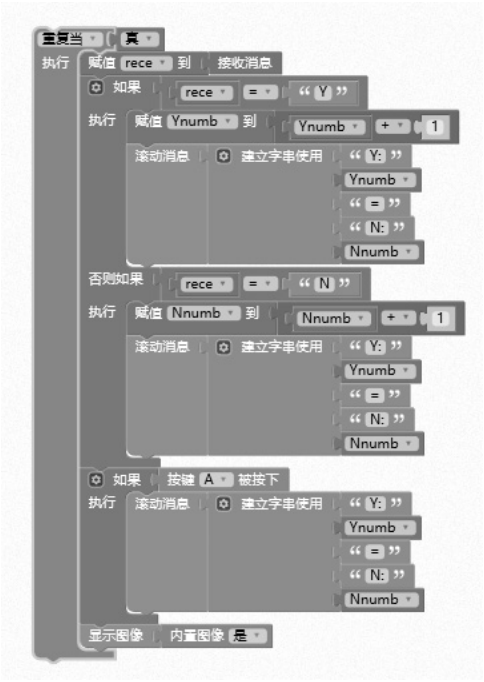


图 7-13　建立无限循环

完整的接收机拼插编程实现程序如图 7-14 所示。

图 7-14　完整的接收机拼插编程实现程序

（2）代码编程

具体解释见代码注释部分。

```
import radio
from microbit import *
```

```
    Ynumb = 0    #定义 Ynumb 并初始化
    Nnumb = 0    #定义 Nnumb 并初始化
    radio.on()     #打开蓝牙
    radio.config(length=32,queue=3,channel=7,power=5,
data_rate=radio.RATE_1MBIT)    #配置蓝牙参数
    while True:
        rece = radio.receive()   #接收数据
        if rece == 'Y':         #如果收到“Y”，那么“同意”人数加 1，并显示结果
             Ynumb = Ynumb + 1
             display.scroll(('').join([str(x) for x in ['Y:', Ynumb, '=',
'N:', Nnumb]]))
        elif rece == 'N':    #如果收到“N”，那么“反对”人数加 1，并显示结果
             Nnumb = Nnumb + 1
             display.scroll(('').join([str(x2) for x2 in ['Y:', Ynumb, '=',
'N:', Nnumb]]))
        if button_a.is_pressed():     #如果按下 A 键，那么显示当前结果
             display.scroll(('').join([str(x3) for x3 in ['Y:', Ynumb, '=',
'N:', Nnumb]]))
        display.show(Image.YES)
```

【思考】

从上面的程序可以看出，在两个条件（如果收到“Y”和如果收到“N”）程序代码基本一样，只是参数发生了改变，那么是不是可以考虑用函数来实现呢？

如果把“广播功率”改为 0 会怎样？“广播功率”最大是多少？

7.3.3　分享代码

你对上面制作的无线投票器还满意吧，有没有分享给朋友们，让他们来点赞？其实分享方法很简单，步骤如下：

① 点击编程界面右上方的“分享”，如图 7-15 所示。

图 7-15　分享代码

② 输入密码，点击“创建链接”按钮，如图 7-16 所示。

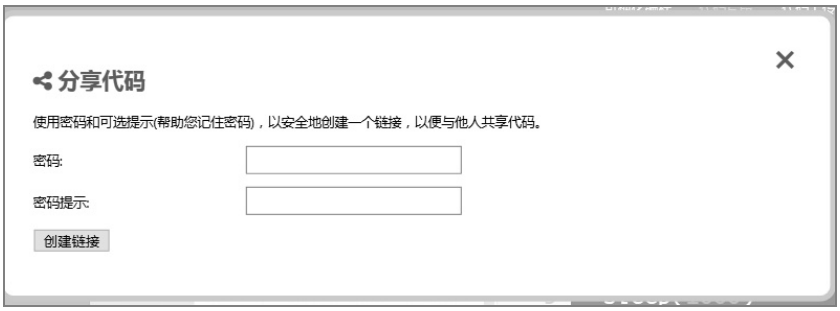


图 7-16　设置密码

③ 生成 HTTP 链接，复制这个 HTTP 链接，发送给你的朋友们，如图 7-17 所示。

图 7-17　生成 HTTP 链接

7.4　知识要点

7.4.1　拼插编程

【掌握】

- 逻辑与运算符。
- 函数的用法。

- 无线电基本知识。

7.4.2　代码编程

【掌握】

函数的使用。

第 8 章　指南针

8.1　制作指南针

作为中国古代四大发明之一，指南针的发明对人类的科学技术和文明的发展起到了不可估量的作用。在中国古代，指南针主要应用于祭祀、礼仪、军事和占卜中来确定方位，现今应用于军事、导航等多领域进行方向确认。指南针又称司南，主要组成部分是一根装在轴上的磁针，磁针在天然地磁场的作用下可以自由转动并保持在磁子午线的切线方向上，磁针的北极指向地理的北极，利用这一性能可以辨别方向。

现在，我们应用 TurnipBit 板子上的磁敏传感器来制作一个指南针。磁敏传感器是把磁场、电流、应力应变、温度、光等外界因素引起的敏感元件磁性能变化转换成电信号，以此来检测相应物理量的器件。在生活中用到磁敏传感器的地方比比皆是，如指南针、电脑硬盘、家用电器等。

所需器材：

- TurnipBit 开发板一块
- 下载数据线一条
- 接入互联网的电脑一台（推荐使用 Google Chrome 或者 Firefox 浏览器）

8.2　确定南方在哪里

8.2.1　学会使用指南针

指南针在使用前必须要校正，罗盘上从 0 到 360°范围内的整数表示顺时针方向的角度，正北方向为 0。根据这一点，显然正南方向为 180°。为了便于调节，我们将 180°±30°都当成南方，也就是 150°~210°都指南方。

① 制作指南针的底座。取一个空纸盒，将 TurnipBit 的电源线从纸盒中间穿出来，在纸盒上面画出南方，其他方向也就知道了，如图 8-1 所示。

图 8-1　指南针底座

放置 TurnipBit，注意放置方向，如图 8-2 所示。

② 道具做好了，接下来开始拼插编程。校正的方法是使用“磁敏传感器”中的“校正指南针”，打开模块化拼插编程界面，将“校正指南针”拖到模块编辑区，右侧是相应的代码，如图 8-3 所示。

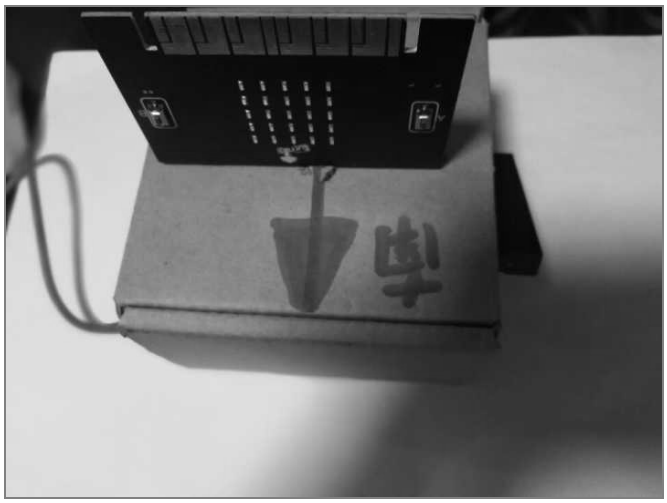


图 8-2　放置 TurnipBit

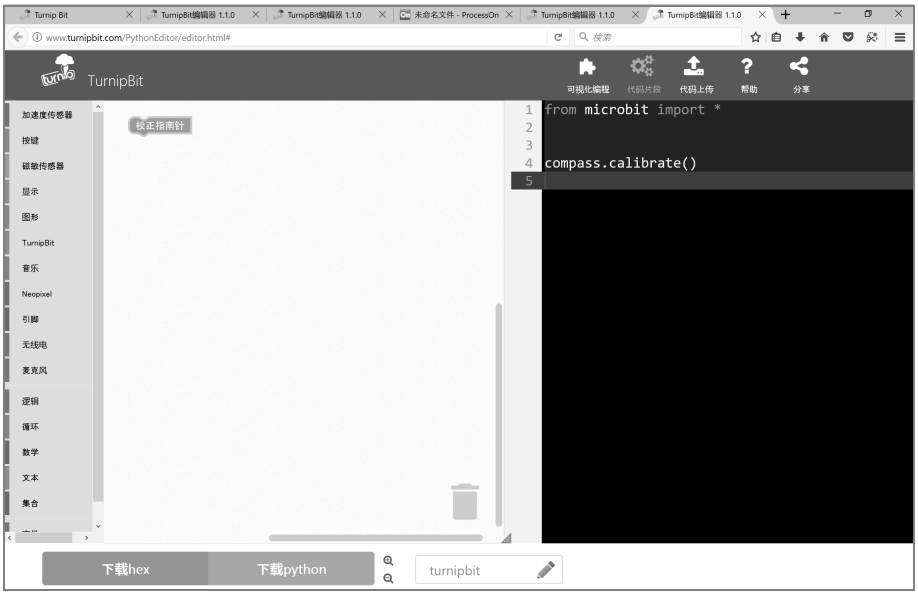


图 8-3　校正指南针

③ 读取指南针的方向，将读取值赋给变量 header。如果 header 的值在 150° 和 210° 之间，则说明当前 LED 屏面对的方向为“南”，我们用“S”表示，如图 8-4 所示。

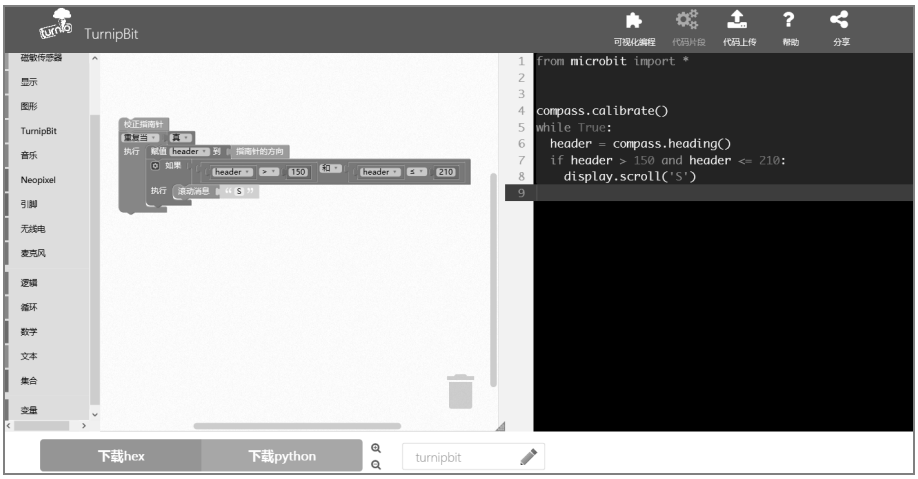


图 8-4　判断方向

保持 TurnipBit 程序名不变，点击“下载 hex”按钮将程序保存到电脑里。将所保存的 HEX 文件拖入 TURNIPBIT 磁盘中，我们会看到 TurnipBit 板子上的灯在闪烁，说明正在下载到控制板中。下载成功后，程序会正常运行，效果如图 8-5 所示。

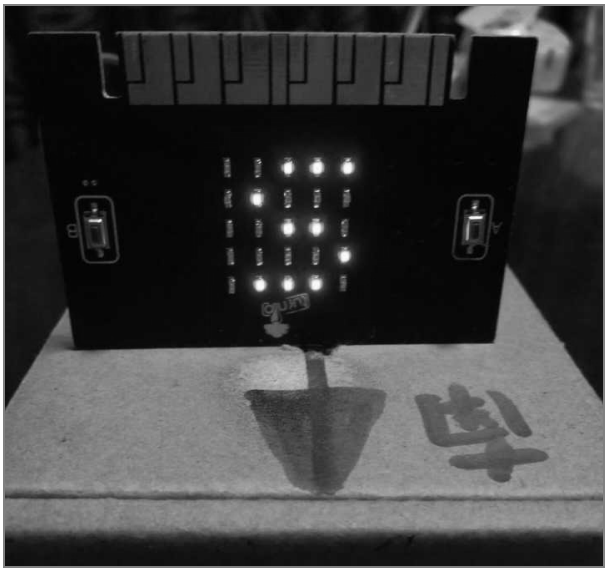


图 8-5　指南针效果图

【思考】

面南背北，左东右西，是什么意思？根据这个能定位出其他几个方向吗？哪个方向是北方？

8.2.2 显示每个方向的指南针

东、南、西、北对应的英文单词分别是 East、South、West、North，因此 E 代表东方，S 代表南方，W 代表西方，N 代表北方。

① 南方是 150°~210°，那么东方、西方和北方分别是多少度呢？

东方是 60°~120°；西方是 240°~300°；北方是小于 30°或者大于 330°，也就是 330°~360°或者 0°~30°。

② 使用学习过的分支语句就能实现。如图 8-6 所示，点击“如果”左侧的齿轮图标，下方会出现“否则如果”，按照图中所示进行拖曳。

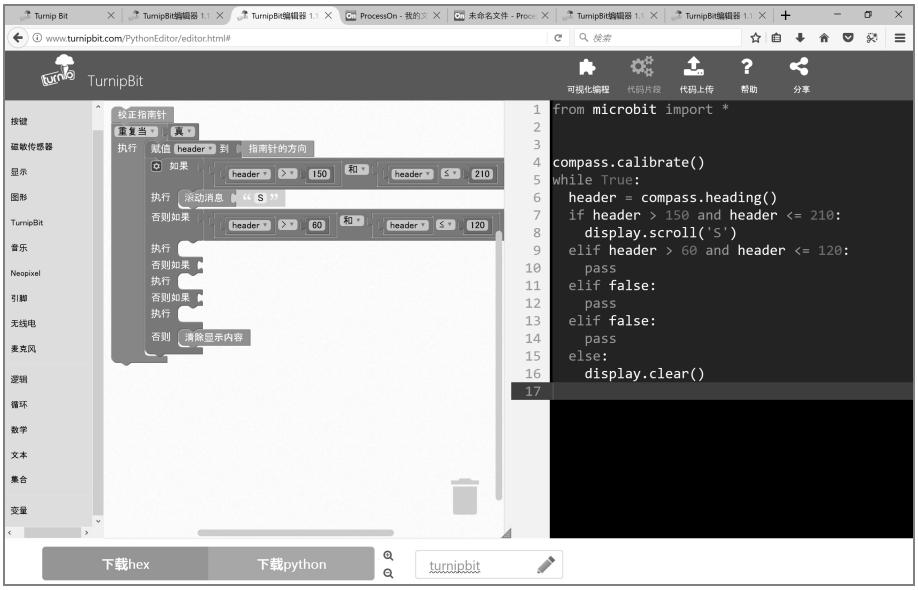


图 8-6　显示方向

③ 在四个“如果”条件中，分别设定为不同的方向，如图 8-7 所示。

条件一：如果 header 在 60°~120°之间，表示当前方向为东，显示“E”。

条件二：如果 header 在 150°~210°之间，表示当前方向为南，显示“S”。

条件三：如果 header 在 240°~300°之间，表示当前方向为西，显示“W”。

条件四：如果 header 小于或等于 30°或者大于 330°，表示当前方向为北，显示“N”。

当然，你也可以自己调整上面的度数，使得方向测量精度更高。

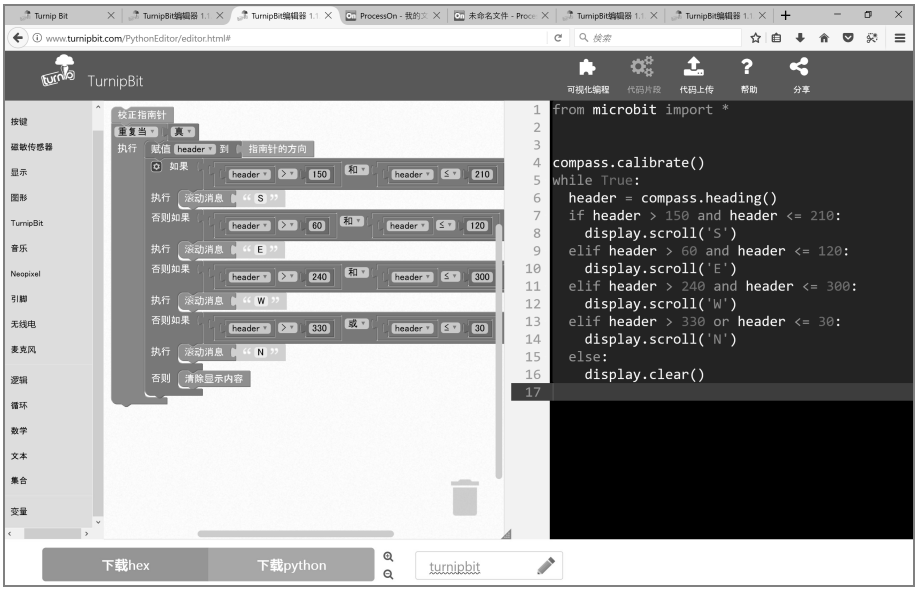


图 8-7　显示当前方向代码

接下来运行程序看看效果如何。保存 HEX 文件，并把该 HEX 文件拖入 TURNIPBIT 磁盘中，我们会看到 TurnipBit 板子上的灯在闪烁，说明正在下载到控制板中。下载成功后，程序会正常运行。

【思考】

对北方的判断为什么要使用大于 330°或者小于 30°？还有别的方法吗？

真实的指南针正北方向都是 0，根据计算得到的北方与真正的北方偏差 5°，那么真正的北方应该如何校正呢？

8.3 指南针流程图

8.3.1 指南针的模糊概念

实际的指南针要复杂得多，因为磁场南极（S）在地理北极的附近，有一定的误差，本试验采取非精确指向北方，也就是说，我们所制作的指南针的北方与地理北极方向偏差 30° 以内时，都当作北方。

8.3.2 绘制流程图

根据前面介绍的拼插编程，我们很容易写出指南针的伪代码，具体如下：

```
Begin（算法开始）
校正指南针
重复当真(循环体 开始)
取得指南针的方向 给 变量 header
If 大于 150° 并且小于或等于 210°
   判定为南方，滚动显示“S”
Elseif 大于 60° 并且小于或等于 120°
   判定为东方，滚动显示“E”
Elseif 大于 240° 并且小于或等于 300°
   判定为西方，滚动显示“W”
Elseif 大于 330° 或者小于或等于 30°
   判定为南方，滚动显示“S”
Else
   清空屏幕
重复当真(循环体 结束)
End （算法结束）
```

根据伪代码绘制流程图，如图 8-8 所示。

【思考】

重复当真还可以使用磁敏传感器块中的哪个插？

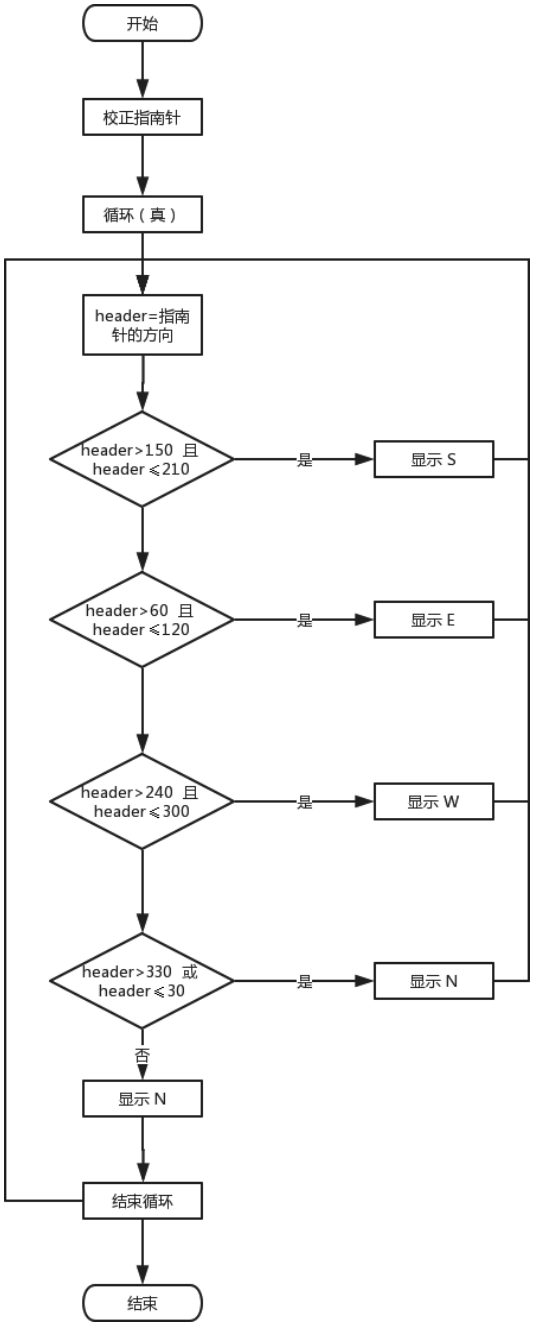

开始
校正指南针
循环（真）
header=指南针的方向
header>150 且 header⩽210
是
显示 S
header>60 且 header⩽120
是
显示 E
header>240 且 header⩽300
是
显示 W
header>330 或 header⩽30
是
显示 N
否
显示 N
结束循环
结束

图 8-8　指南针流程图

8.4 知识要点

8.4.1 拼插编程

【掌握】

- 指南针校正。
- 指南针的方向是 0~360°。
- 在“如果”中添加“否则如果”。

【理解】

- 指南针的模糊概念。
- 0（=360°）临界值正负的判断方法。

8.4.2 代码编程

【掌握】

- 流程图中“变量”的画法。
- 流程图中“循环”的用法。

【理解】

- 取得指南针的方向为什么要放到循环体内。
- “清除显示内容”放到“否则”中有什么作用。

第 9 章　简易的 MP3 播放器

9.1　如何播放美妙的音乐

当我们想听美妙的音乐的时候，就需要通过音乐播放器来播放各种各样的音乐文件。常见的音乐文件格式有 MP3、WMA、WAV 等。一般情况下，播放器都支持一种或多种格式的音乐文件。就像我们平常见到的 MP3 播放器，它就是支持播放 MP3 格式的音乐播放器。MP3 播放器其实就是一个功能特定的“微型电脑”。在 MP3 播放器小小的机身里，拥有 MP3 播放器存储器（存储音乐文件）、MP3 播放器中央处理器（微控制器）等。微处理器是播放器的“大脑”，用来接受用户所选择的播放控制，比如播放、暂停、下一首、上一首等。

现在，我们就用 TurnipBit 来做一个简易的 MP3 播放器，播放 TurnipBit 内置的音乐文件，让我们跟着音乐一起唱起来吧！

所需器材：

- TurnipBit 开发板一块
- 下载数据线一条
- 有线耳机一个
- 接入互联网的电脑一台（推荐使用 Google Chrome 或者 Firefox 浏览器）

9.2 播放音乐

9.2.1 一首音乐循环播放

1. 拼插编程

① 打开 TurnipBit 编辑器（网址是：http://turnipbit.com/PythonEditor/editor.html），点击“可视化编程”。

② 在左侧的块选择区中找到“音乐”块，用鼠标选中  ，拖曳至模块编辑区，如图 9-1 所示。

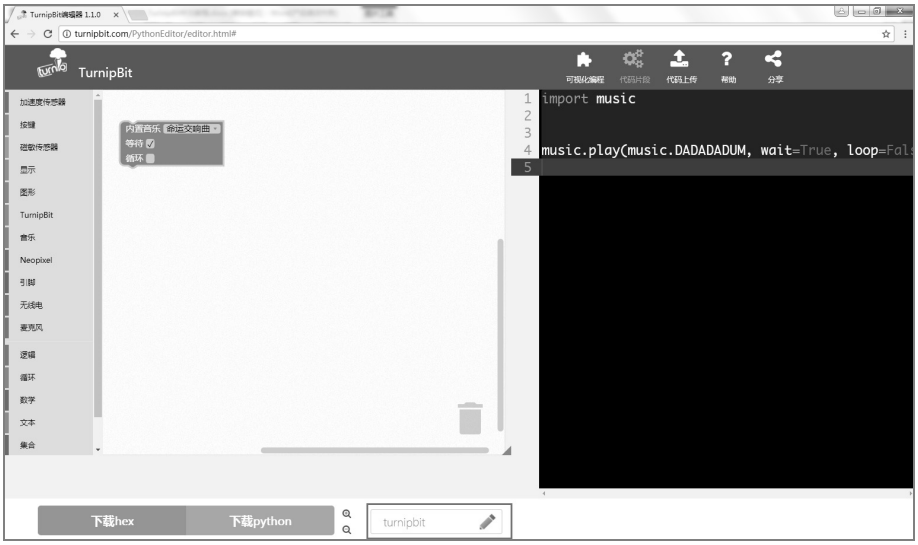


图 9-1 “音乐”块

③ 点击“内置音乐”的下拉框，展开显示 TurnipBit 中所有内置的音乐曲目，如图 9-2 所示。

在这里可以选择一首音乐进行播放，在“内置音乐”的下方有“等待”和“循环”两个选项，其中等待表示音乐播放一遍后停止播放；循环表示音乐会一直不断地播放。

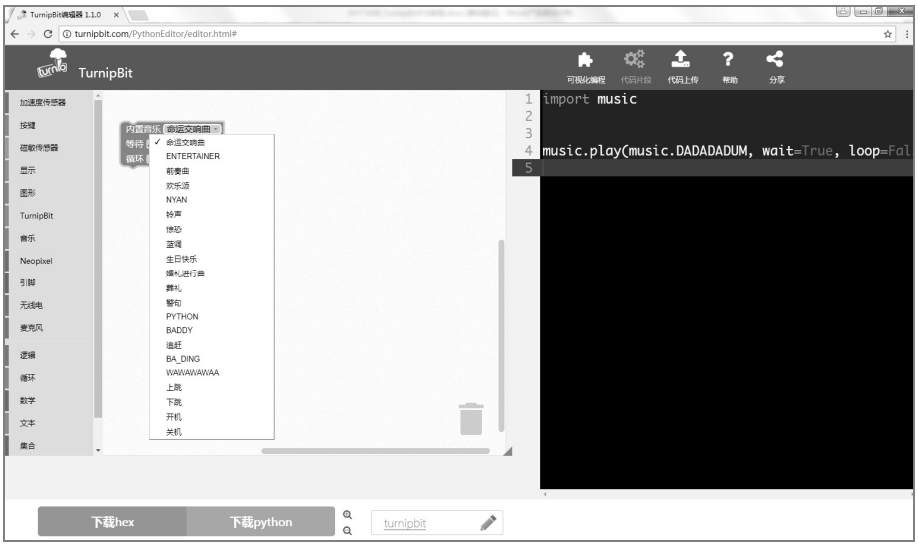


图 9-2　选择内置音乐

本节要实现单曲循环播放的功能，所以这里选择“循环”，如图 9-3 所示。

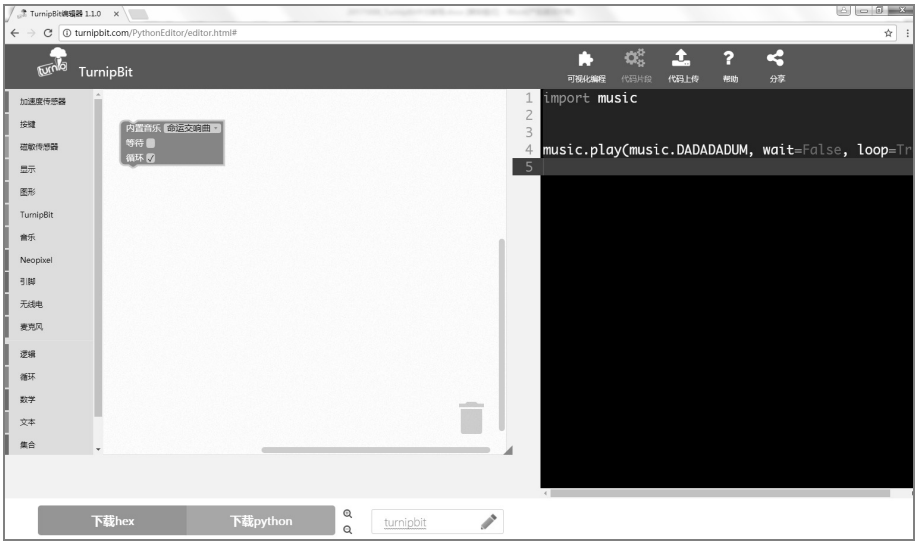


图 9-3 “循环”设计

④ 将程序名修改成 turnipbit-music，点击“下载 hex”按钮，将程序保存到电脑里，如图 9-4 所示。将所保存的 TurnipBit-music.hex 文件拖入 TURNIPBIT 磁

盘中，我们会看到 TurnipBit 板子上的灯在闪烁，说明正在下载到控制板中。下载成功后，将耳机插入 TurnipBit 的耳机插孔中，我们就可以听到美妙的音乐了。

图 9-4　保存文件

2．代码分析

在听音乐的同时，我们观察一下右侧的代码显示区。

（1）引用音乐库

```
import music
```

这行代码成功地将 music 库引入程序中，此后在程序中就可以直接调用 music 库里的方法了，如调用 play()方法，就用 music.play()。

细心的你可能会问，前面不是还提到过 from 引入库的方法吗？如果用 from 这里该怎么写呢？

```
from music import *
```

如果使用这条语句来引入 music 库，那么在程序中调用库里的方法时，就不需要再写库名了，如调用 play()方法，就可以直接用 play()。

（2）播放音乐的 play()方法

从程序来看，播放音乐使用的是 music 库中的 play()方法，其具体的语法格式为：

```
music.play(music.DADADADUM, wait=True, loop=False)
```

music.play()方法有 3 个参数，分别是要播放的音乐名称（如 music.DADADADUM）、设置的等待模式（wait=True）、设置的循环模式（loop=False）。若选中等待模式，则 wait=True；反之，wait=False。若选中循环模式，则 loop= True；反之，loop =False。

TurnipBit 内置了 20 余首音乐，具体列表如下：

- music.DADADADUM——命运交响曲
- music.ENTERTAINER——ENTERTAINER

- music.PRELUDE——前奏曲
- music.ODE——欢乐颂
- music.NYAN——NYAN
- music.RINGTONE——铃声
- music.FUNK——惊恐
- music.BLUES——蓝调
- music.BIRTHDAY——生日快乐
- music.WEDDING——婚礼进行曲
- music.FUNERAL——葬礼
- music.PUNCHLINE——警句
- music.PYTHON——PYTHON
- music.BADDY——BADDY
- music.CHASE——追赶
- music.BA_DING——BA_DING
- music.WAWAWAWAA——WAWAWAWAA
- music.JUMP_UP——上跳
- music.JUMP_DOWN——下跳
- music.POWER_UP——开机
- music.POWER_DOWN——关机

【思考】

是否可以通过按键来控制播放和停止呢？

9.2.2　TurnipBit 音乐播放器拼插编程

上一节中我们学习了如何实现一首音乐的循环播放，那么如何制作一个能播

放多首音乐的音乐播放器呢？本书我们就来学习使用 TurnipBit 开发板制作音乐播放器，其主要功能是实现多首音乐的播放，并且能通过按键来控制切换。

我们先用伪代码来分析程序的逻辑。首先假设有 4 首（数量可自定）不同的音乐，当按键 A 被按下时，切换到下一首音乐播放；当按键 B 被按下时，切换到上一首音乐播放。最后处理一下两头的音乐，假设当前正在播放第四首音乐，则按下按键 A 时，切换到第一首音乐播放；当前正在播放第一首音乐，则按下按键 B 时，切换到第四首音乐播放。于是，使用伪代码可以表示为：

① 准备 4 首不同的内置音乐。

② 新建一个变量 *n*，用于存放当前要播放的音乐的序号。若要播放第一首音乐，则 *n*=1。依此类推（初始值为 1）。

③ 添加一个永远不停止的循环（死循环）。

④ 在死循环中，监测按键 A、B 的状态，判断它们是否被按下。

⑤ 根据按键 A、B 的状态和变量 *n* 当前的值计算出接下来要播放的音乐的序号，并赋值给变量 *n*。

n 的值与播放音乐的对应关系如下（音乐可自定）：

- 当 *n*=1 时，命运交响曲
- 当 *n*=2 时，欢乐颂
- 当 *n*=3 时，生日快乐
- 当 *n*=4 时，婚礼进行曲

画出流程图，如图 9-5 所示。

根据流程图，我们开始编程。

① 打开 TurnipBit 编辑器（网址为：http://turnipbit.com/PythonEditor/editor.html），点击“可视化编程”，如图 9-6 所示。

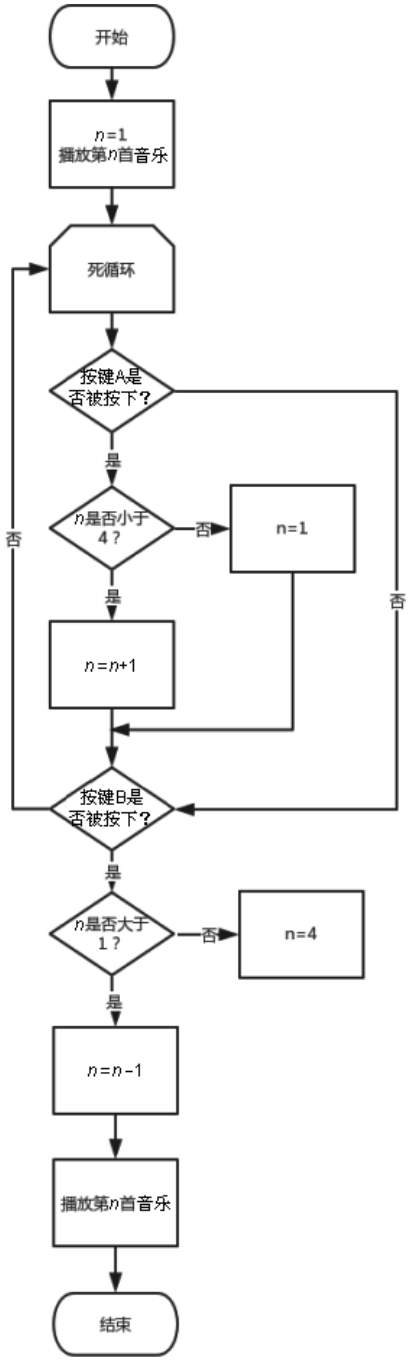


图 9-5　音乐播放器流程图

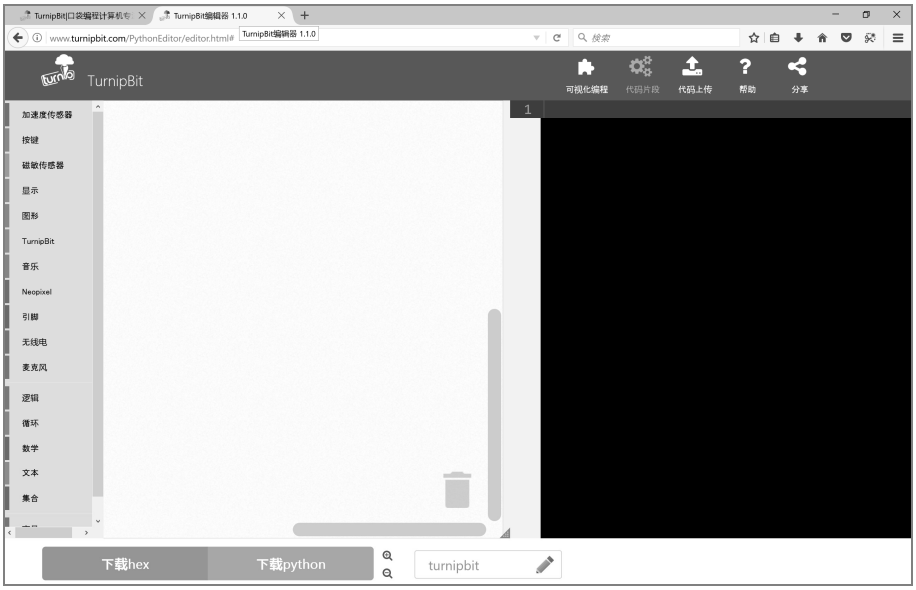


图 9-6　编程界面

② 根据在第 4 章中学习的创建变量的知识，建立一个变量 *n*，用来记录当前音乐的序号，并初始化 *n* 的值为 1，如图 9-7 所示。

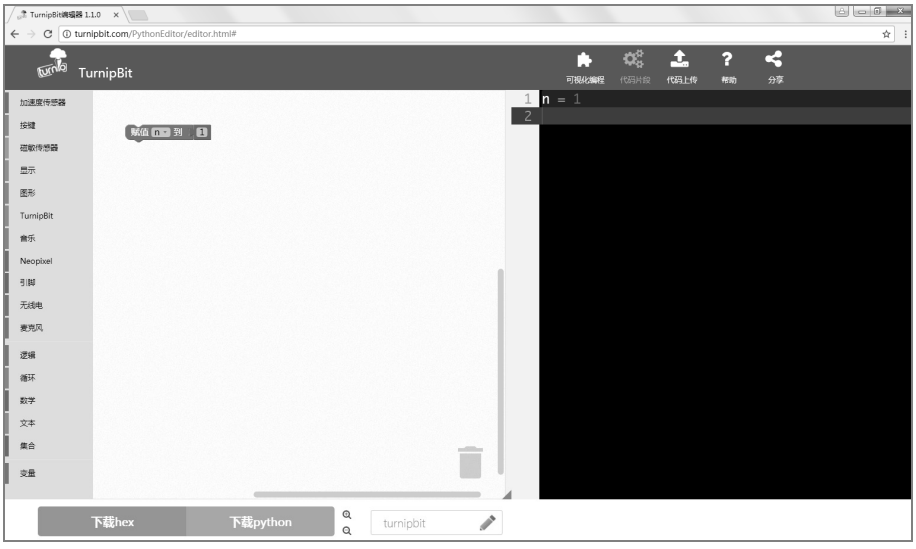


图 9-7　新建变量 *n*

③ 在左侧的块选择区中找到“循环”块，用鼠标选中  ，拖曳至模块编辑区，与“赋值 *n* 到”拼接，如图 9-8 所示。

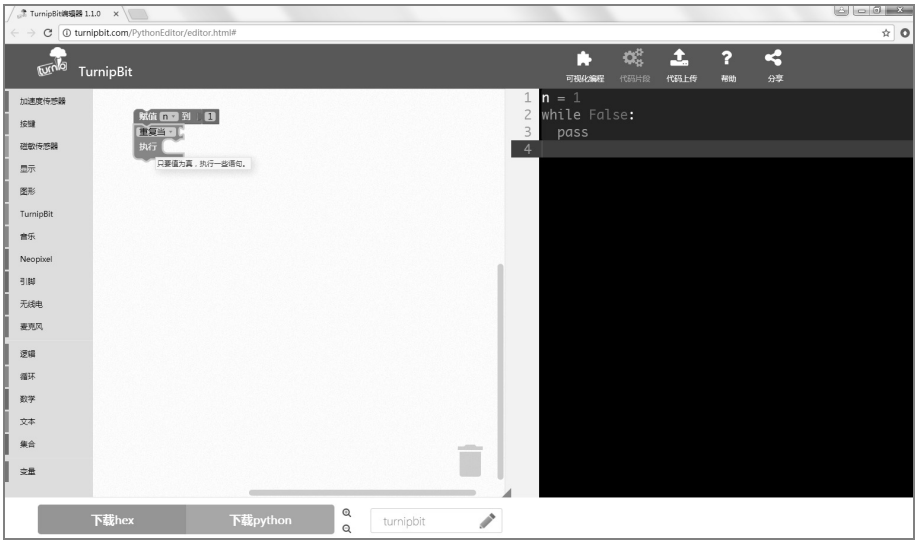


图 9-8　添加循环

④ 在左侧的块选择区中找到“逻辑”块，用鼠标选中  ，实现无限循环。

⑤ 在左侧的块选择区中找到“逻辑”块，用鼠标选中  。在“如果 执行 ”条件中加入“按键 A 被按下”。同时，复制整个“如果 执行 ”模块，将“按键 A 被按下”改为“按键 B 被按下”，如图 9-9 所示。

图 9-9　加入判断按键 A、B 是否被按下

⑥ 在“如果按键 A 被按下”条件中添加“如果 执行 ”模块。根据流程图可以看出，当按键 A 被按下时，需要判断 *n* 是否小于 4。如果 $n<4$，那么 $n=n+1$，否则 $n=1$。所以，在这里的“如果 执行 ”模块中需要添加“否则”，变为“如果

执行 否则 ”。具体如图 9-10 所示。

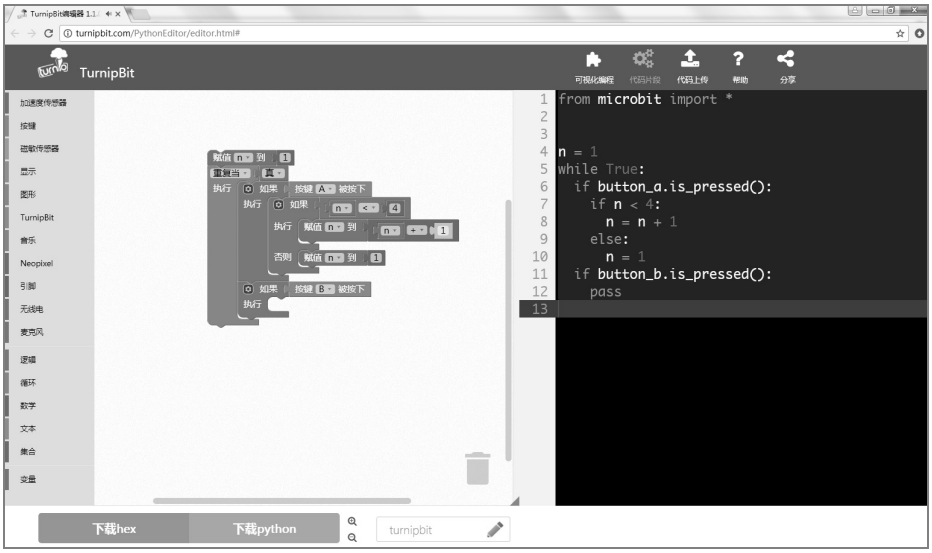


图 9-10 按下 A 键播放下一首音乐

⑦ 参考第 6 步和流程图，完成当按键 B 被按下时的程序编写。当按键 B 被按下时，判断 *n* 是否大于 1。如果 *n*>1，那么 *n*=*n*-1，否则 *n*=4，具体如图 9-11 所示。

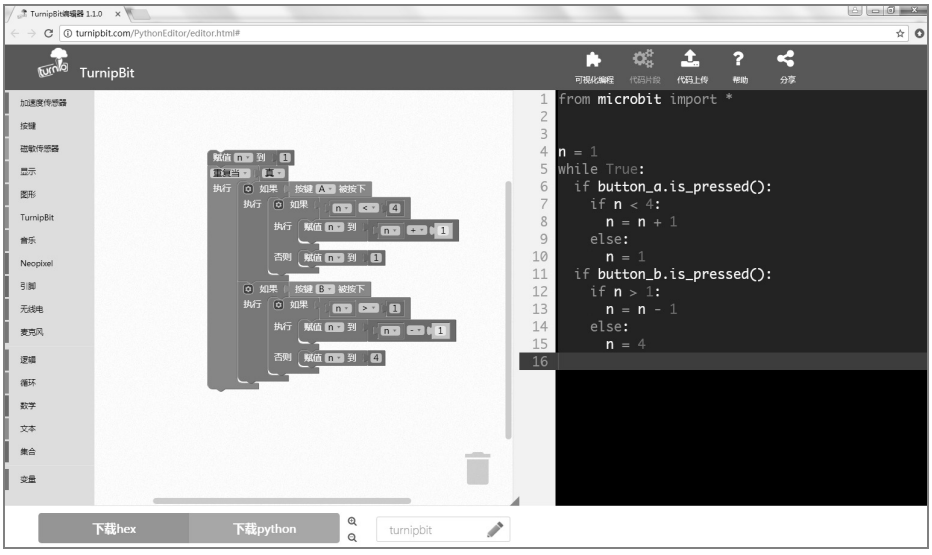


图 9-11 按下 B 键播放上一首音乐

⑧ 判断 n 的值播放相应的音乐，如图 9-12 所示。

图 9-12　根据序号播放音乐

最后，得到完整的代码，如图 9-13 所示。

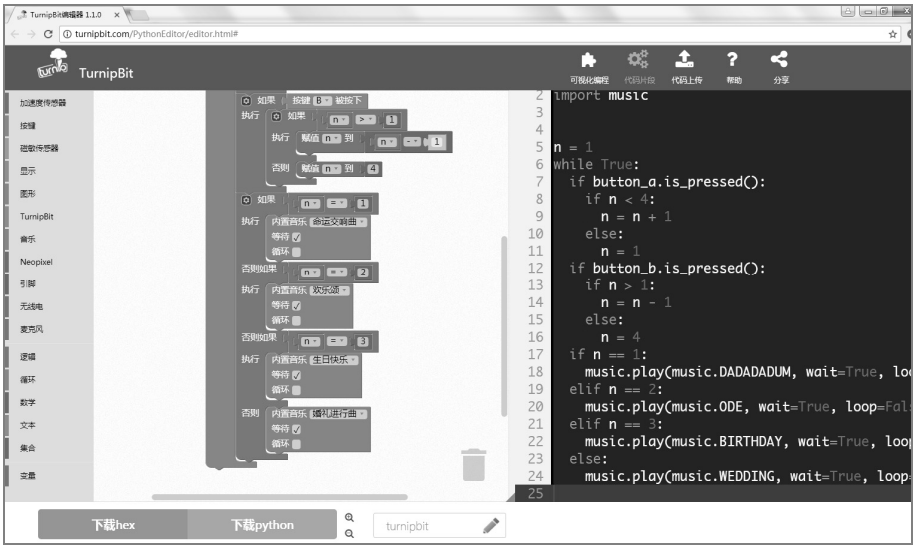


图 9-13　音乐播放器代码

如果把这段代码拷贝到 TurnipBit 中，就会发现音乐不停地播放，并且无法检测按键 A、B 的状态，切换音乐更是无法实现。这是什么原因导致的呢？

进一步分析上面的程序，发现在播放音乐的过程中，程序无法检测按下按键的动作，也就是说，在音乐播放过程中按键是不起作用的。当程序运行后，*n* 为 1，开始播放音乐。播放完一首音乐后，由于在 while 循环内，时间间隔很短，无法判断按键，导致 *n* 的值未发生变化，继续播放，按键不起作用。

为了解决这一问题，需要再添加一个变量 k，其初始值为 True（真）。当 k 为 True 时，才会执行播放音乐的结构体；播放完音乐后，将 k 赋值为 False（假），等待按键切换音乐。当我们按下按键 A 或 B 后，将 k 重新赋值为 True，说明进行了切换操作，需要播放新的音乐。最后，修改后的可视化编程拼插块如图 9-14 所示。

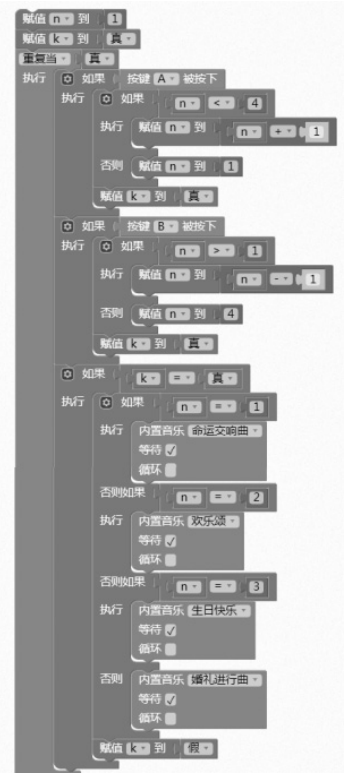


图 9-14　修改后的可视化编程拼插块

⑨ 将程序名修改成 turnipbit-player，点击“下载 hex”按钮，将程序保存到电脑里，如图 9-15 所示。把所保存的 turnipbit-player.hex 文件拖入 TURNIPBIT 磁盘中，我们会看到 TurnipBit 板子上的灯在闪烁，说明正在下载到控制板中。下载成功后，我们首先听到的是《命运交响曲》的音乐，然后通过按键 A、B 就可自由地进行音乐切换了。

图 9-15　保存文件

注意：需要等待音乐播放完毕后，此时 k 为 False，等待按键时，按键切换才有效。

9.2.3　音乐播放器代码分析

TurnipBit 音乐播放器的整体代码如下：

```
from microbit import *
import music
n = 1
k = True
while True:
    if button_a.is_pressed():
        if n < 4:
            n = n + 1
        else:
            n = 1
        k = True
    if button_b.is_pressed():
        if n > 1:
            n = n - 1
        else:
            n = 4
        k = True
    if k == True:
        if n == 1:
```

```
            music.play(music.DADADADUM, wait=True, loop=False)
        elif n == 2:
            music.play(music.ODE, wait=True, loop=False)
        elif n == 3:
            music.play(music.BIRTHDAY, wait=True, loop=False)
        else:
            music.play(music.WEDDING, wait=True, loop=False)
        k = False
```

在这段代码中，我们主要用到了以下知识点。

1．死循环

在程序设计中，死循环就是一个自身无法终止的程序，会一直无休止地运行。本例使用死循环，就是为了能实时监测按键 A、B 的状态。如果不使用死循环，程序只会运行一次，无法实时获取到按键 A、B 的状态。

```
while True:
    代码块
```

2．按键操作

在左侧的块选择区中找到“按键”块，可以看到关于按键的操作有 3 种，它们的含义不同，用到的方法也不同。

（1）按键 A/B 被按下

```
button_a.is_pressed()
#按下按键 A 时，该方法会返回 True，否则返回 False
```

（2）按键 A/B 曾经按下

```
button_a.was_pressed()
#按下按键 A 抬起松开后，该方法会返回 True，否则返回 False
```

（3）按键 A/B 按下过的次数

```
button_a.get_presses()
    #该方法返回按键 A 连续快速按下的总次数，间隔过长会清零
```

【思考】

除了播放内置的音乐，是否可以播放自己编写的音乐呢？

9.3　TurnipBit 播放自定义音乐

上一节介绍的是用 TurnipBit 播放内置的音乐，本节介绍让 TurnipBit 播放自定义音乐。

9.3.1　TurnipBit 播放音乐的方法

音乐乐谱主要分音调、节拍等，如果我们能把音调及节拍呈现出来，那么基本就能表现出音乐的效果。用 TurnipBit 来播放音乐，其实就是简单地呈现音调及节拍，这就是我们用耳机听到的 TurnipBit 音乐并不像 MP3 等播放器播放的音乐那么动听的原因。

1．音调

声音的高低叫作音调。在简谱中，用于表示音的高低及相互关系的基本符号为 7 个阿拉伯数字，即 1、2、3、4、5、6、7，唱作 do、re、mi、fa、sol、la、si，称为唱名。每个唱名又对应一个音名，分别为 C、D、E、F、G、A、B，如表 9-1 所示。

表 9-1　音调

简谱	1	2	3	4	5	6	7
唱名	do	re	mi	fa	sol	la	si
音名	C	D	E	F	G	A	B
中文	哆	来	咪	发	唆	啦	西

音调又分为低音、中音和高音，其区别就是频率不同。在这里我们先搞清楚各音调的频率，具体如表 9-2、表 9-3 和表 9-4 所示：

表 9-2　低音频率表

音调音符	$1_{\#}$	$2_{\#}$	$3_{\#}$	$4_{\#}$	$5_{\#}$	$6_{\#}$	$7_{\#}$
A	221	248	278	294	330	371	416
B	248	278	294	330	371	416	467
C	131	147	165	175	196	221	248
D	147	165	175	196	221	248	278
E	165	175	196	221	248	278	312
F	175	196	221	234	262	294	330
G	196	221	234	262	294	330	371

表 9-3　中音频率表

音调音符	1	2	3	4	5	6	7
A	441	495	556	589	661	742	833
B	495	556	624	661	742	833	935
C	262	294	330	350	393	441	495
D	294	330	350	393	441	495	556
E	330	350	393	441	495	556	624
F	350	393	441	495	556	624	661
G	393	441	495	556	624	661	742

表 9-4　高音频率表

音调音符	$1^{\#}$	$2^{\#}$	$3^{\#}$	$4^{\#}$	$5^{\#}$	$6^{\#}$	$7^{\#}$
A	882	990	1112	1178	1322	1484	1665
B	990	1112	1178	1322	1484	1665	1869
C	525	589	661	700	786	882	990
D	589	661	700	786	882	990	1112
E	661	700	786	882	990	1112	1248
F	700	786	882	935	1049	1178	1322
G	786	882	990	1049	1178	1322	1484

2．节拍

有了音调就差节拍了，节拍就是控制着音符演奏的时间长短。每个音符都会播放一定的时间，这样才能构成一首优美的曲子，而不是生硬地一个调把所有音符一股脑都播放出来。音符节奏分为一拍、半拍、四分之一拍、八分之一拍等。如果我们规定音符的时间，1 拍为 4，那么半拍就为 2，四分之一拍为 1，依此类推。最后，我们把每个音符赋予这样的节拍并播放出来，就形成了音乐。

那么，如何让 TurnipBit 演奏呢？music 库中的 play()方法是可以直接演奏音调的。比如要演奏“哆”音，音高为 4，时长为 4，则表示为 C4:4；演奏“来”音，音高为 4，时长为 4，则表示为 D4:4。于是，具体的播放代码为：

```
import music
tune = ["C4:4", "D4:4"]
music.play(tune)
```

9.3.2　播放自定义音乐实例

① 打开 TurnipBit 编辑器（网址为 http://turnipbit.com/PythonEditor/editor.html），这次需要使用代码编程界面完成自定义音乐的播放，将如下代码复制到代码器中，如图 9-16 所示。

```
    import music
    tune = ["C4:4", "D4:4", "E4:4", "C4:4", "C4:4", "D4:4", "E4:4", "C4:4",
"E4:4", "F4:4", "G4:8", "E4:4", "F4:4", "G4:8"]
    music.play(tune)
```

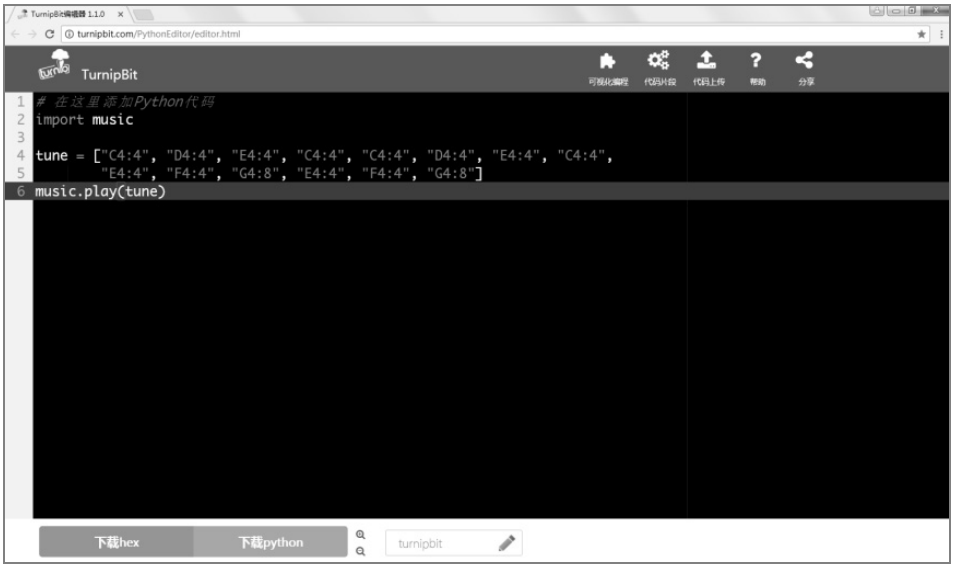


图 9-16　播放自定义音乐的代码

② 将程序名修改成 turnipbit-diymusic，点击“下载 hex”按钮，将程序保存到电脑里，如图 9-17 所示。把所保存的 turnipbit-diymusi.hex 文件拖入 TURNIPBIT 磁盘中，我们会看到 TurnipBit 板子上的灯在闪烁，说明正在下载到控制板中。下载成功后，插上耳机我们就可以听到自定义的音乐了。

图 9-17　保存文件

9.3.3 播放自定义音乐代码分析

你是不是已经听出来上面例子播放的是什么音乐了？下面我们以《小星星》（简谱如图 9-18 所示）为例来分析代码。

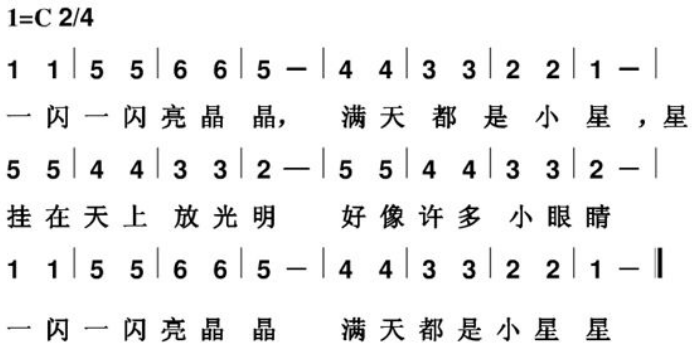


图 9-18 《小星星》简谱

从简谱来看，该音乐为 C 调，节拍为 2/4。第一个音符为 1，即“哆”。根据前面所讲的知识，用 C4:4 来表示，C 音名对应的简谱符号就是 1，C4 表示标准音高的 C。紧接着冒号右侧的 4 表示发音持续的时间，于是对应形成的 tune 的值为：

```
    tune = ["C4:4", "C4:4", "G4:4", "G4:4", "A4:4", "A4:4", "G4:8", "F4:4",
"F4:4", "E4:4", "E4:4", "D4:4", "D4:4", "C4:8"]
```

完整的代码如下：

```
    import music

    tune = ["C4:4", "C4:4", "G4:4", "G4:4", "A4:4", "A4:4", "G4:8", "F4:4",
"F4:4", "E4:4", "E4:4", "D4:4", "D4:4", "C4:8"]
    music.play(tune)
```

保存程序，并将 HEX 文件拖入 TURNIPBIT 磁盘中，下载成功后，“一闪一闪亮晶晶”熟悉的旋律在耳边响起。

【思考】

试着将《小星星》的音乐全部完成。考虑一下，是否可以设计实现一个电子琴呢？

9.4　知识要点

9.4.1　拼插编程

【掌握】

- 播放指定的内置音乐。
- 学会使用逻辑判断块。
- 对按键状态的判断。

【理解】

- 死循环的概念和应用。
- 按键的 3 种状态。

9.4.2　代码编程

【掌握】

可以自己编写音乐旋律并播放。

【理解】

将简谱转换为 TurnipBit 可播放音乐的原理。

第 10 章　储钱罐

10.1　DIY 储钱罐

储钱罐是我们在日常生活中常见的一样东西。它经常以一头小猪的形象出现，因为猪食量大，体胖身圆，有招财纳福之意。其英文名叫作 The piggy，译为“贪心的小猪猪”。它可以帮助小朋友们养成良好的理财习惯。现在，我们就使用 TurnipBit 制作一个简易的自动计数储钱罐。当我们做完以后，你也许有更多的创意，让这头小猪变得更加智能。

所需器材：

- TurnipBit 开发板一块
- 下载数据线一条
- 纸盒（我们使用了飞机盒）一个
- 导线若干
- 金属夹片两个
- 接入互联网的电脑一台（推荐使用 Google Chrome 或者 Firefox 浏览器）

10.2　绘制储钱罐流程图

我们先来看储钱罐的设计需求。储钱罐以一个盒子或者容器的形式来实现，在盒子或者容器上有一个可以放入硬币的开口，当硬币放入时，储钱罐能显示当

前硬币的个数。如果只放入一元硬币，那么根据个数就能简单地计算出当前存钱的数量。根据这个需求，写出伪代码如下：

```
Begin（算法开始）
定义存钱数的变量
无限循环（while）
判断引脚 0（pin0）是否接通
    存钱数加 1
    延时 2 秒（防止同一枚硬币多次记录）
显示数字
End（算法结束）
```

画出流程图，如图 10-1 所示。

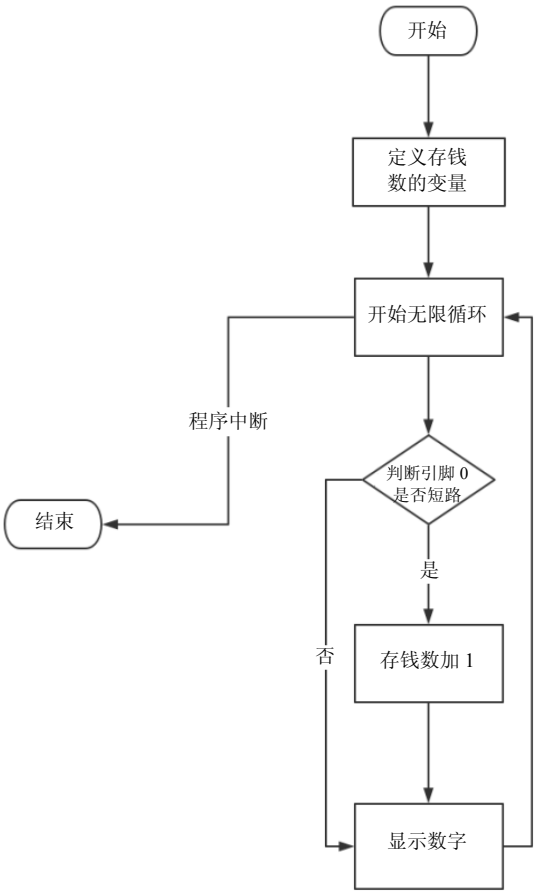


图 10-1　储钱罐流程图

10.3 动手进行拼插编程

10.3.1 实现储钱罐

储钱罐的制作过程分为两个步骤：一是硬件部分的制作；二是软件部分的实现。从这一章开始，我们的实验将基本分为这两步，这也是在进行产品设计和制作过程中必需的两个步骤。

1．硬件制作

① 找一个硬纸盒，并在纸盒顶端做一个开口，大小以正好能放入一元硬币为最佳。

② 在纸盒正面做三个开口，其中中间开口的大小正好能露出显示屏，两边的两个开口为按键，如图 10-2 所示。

图 10-2 纸盒开口示意图

③ 在这个实验中，我们要用到 TurnipBit 的引脚。首先要搞清楚各引脚的含义。从引脚图（见图 10-3）可以看出，TurnipBit 共有 28 个引脚，每个引脚代表不同的含义。这里重点讲解电源引脚和 P 类引脚。电源引脚分为 3.3V 和 GND。3.3V 为电源输入电压，GND 为地，也就相当于电源的负极。P 类引脚又称 GPIO，主要用于通过输入与输出的电平来判断值，比如本例中将使用 P0（PIN0）引脚。我们判断的逻辑是，在没有硬币投入时，P0（PIN0）引脚与 3.3V 电源引脚是断开的，本身是低电平；当有硬币投入时，P0（PIN0）引脚与 3.3V 电源引脚导通，变为高电平，此时通过 P0（PIN0）引脚电压的变化来判断是否有硬币投入。因此，需要通过两条导线将 P0（PIN0）引脚和 3.3V 电源引脚引出，连接到硬币投入口的两边，保证当硬币经过时会接触到这两条导线，从而让 P0（PIN0）引脚与 3.3V 电源引脚连通。导线及 TurnipBit 图如图 10-4 所示。导线安装图如图 10-5 所示。

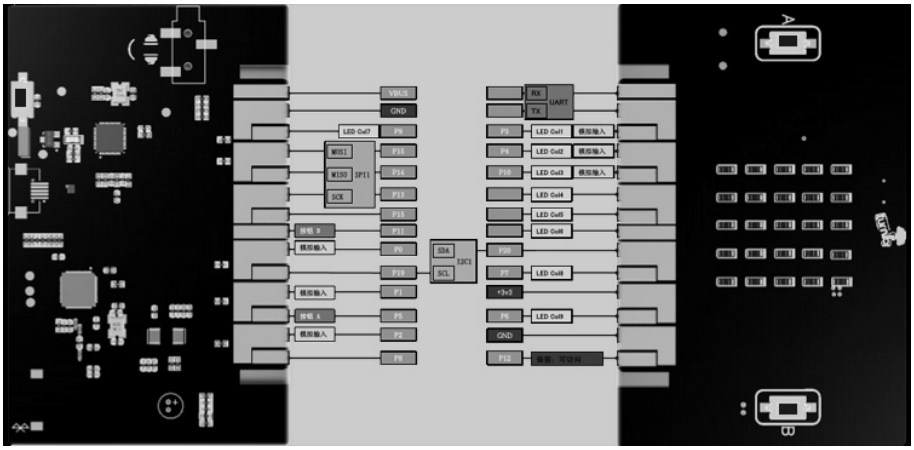


图 10-3　TurnipBit 引脚图

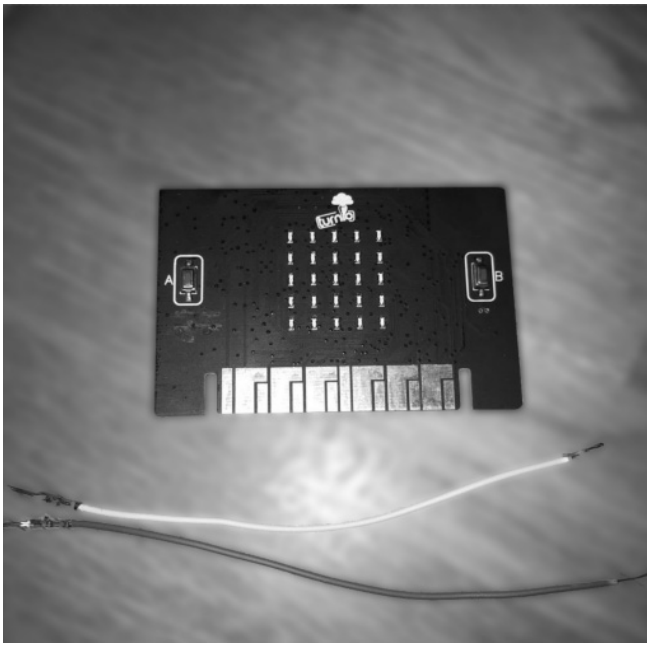


图 10-4　导线及 TurnipBit 图

图 10-5　导线安装图

④ 放入 TurnipBit，贴合正面开口并使 LED 显示屏朝外，将纸盒封装好，如图 10-6 所示。

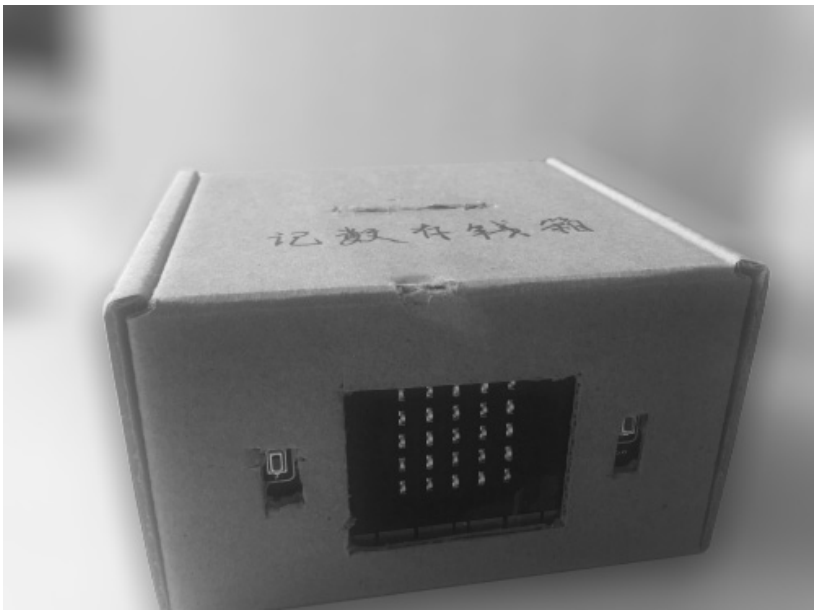


图 10-6　TurnipBit 储钱罐硬件完成图

2．软件实现

根据流程图，首先定义一个数组，用来保存显示图形的代。然后定义一个函数，用来显示图形。在 while 循环中循环扫描两个金属夹片是否短路，如果检测到短路，则延时 2 秒。之所以延时 2 秒，就是为了确保硬币已经落下去；否则，在硬币落下的过程中，可能会多次计数。最后计数加 1，对显示屏上所显示的数值进行相应的调整。

具体的实现过程如下：

① 打开 TurnipBit 官方网站（网址为 http://www.turnipbit.com/），如图 10-7 所示，点击“开始编程”按钮进入编程界面。

② 创建变量 coin_count 用来记录硬币数量；创建变量 pin0 用来存储引脚 0 的值，如图 10-8 所示。

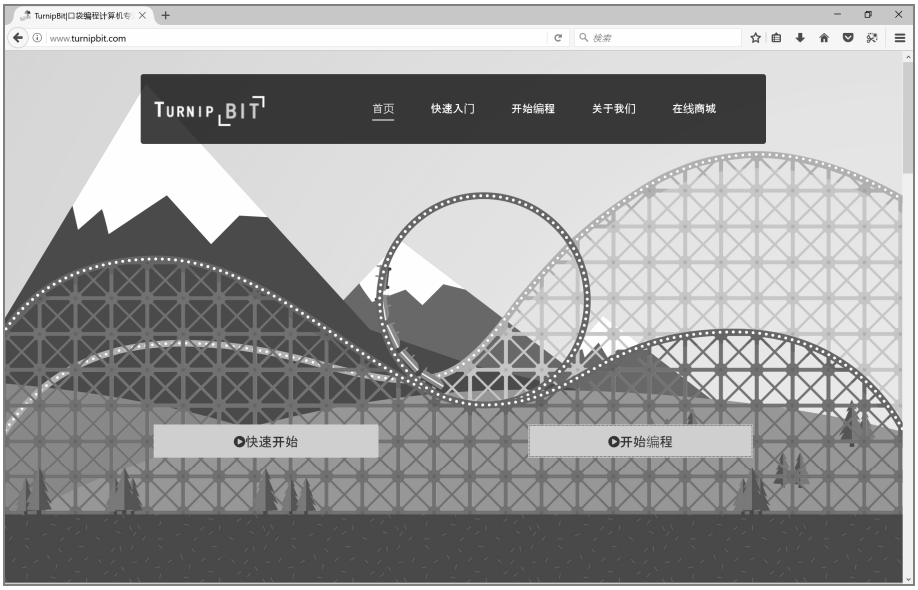


图 10-7　TurnipBit 官方网站

图 10-8　创建变量

③ 在无限循环中，给 pin0 赋值“读取引脚 0 的数字输入”，然后创建“如果执行 ”，在“如果”条件中判断 pin0 是否为 1，如果为 1，则表示是高电平，此时电路是导通的，说明有硬币通过，如图 10-9 所示。

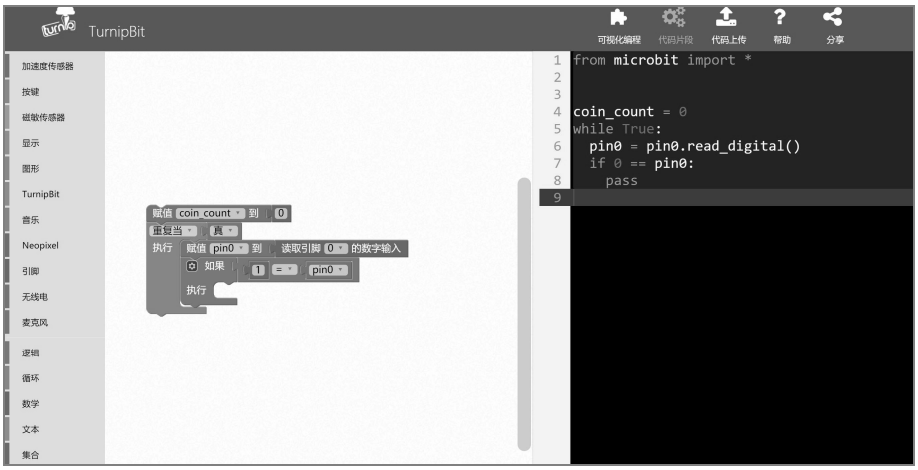


图 10-9　建立循环

④ 若判断为真，则存钱数加 1，并延时 2 秒后显示出存钱数，如图 10-10 所示。

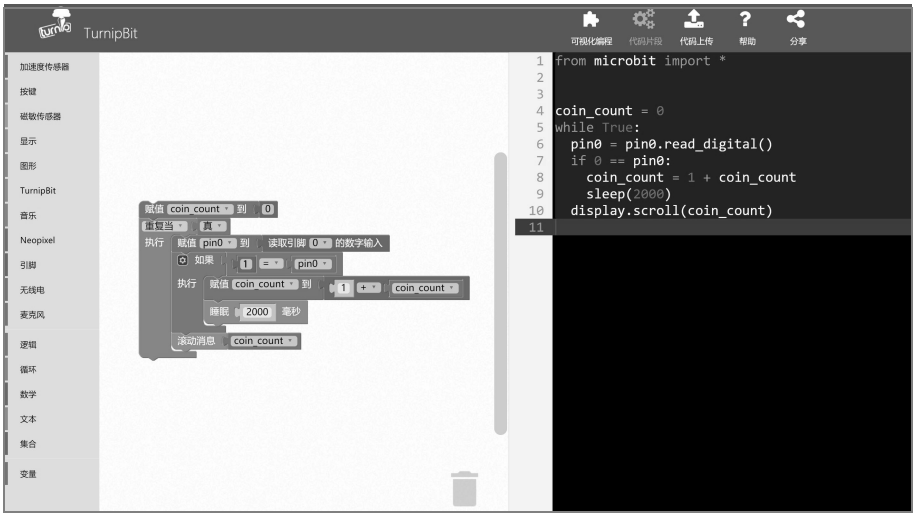


图 10-10　计算存钱数

⑤ 下载 HEX 文件，运行程序看一下效果。

【思考】

我们发现数字是滚动的，那么如何进行修改呢？请根据前面章节中学过的知识自己进行尝试。

10.3.2 进阶实现

硬币本身是凹凸不平的，当有硬币通过时，可能会出现连续导通多次的情况，程序判断有多枚硬币通过，这样就出现了误判。该如何解决这个问题呢？虽然加了延时 2 秒来确保硬币落下后再计数，但这是不是比较好的判断方法呢？其实，在硬件的设计过程中，这种因为抖动所引起的误差是很常见的，解决方法有两种，其中一种方法是通过硬件设计来解决。在这里由于不再对 TurnipBit 硬件进行分析来修正，所以这种方法就不做详细介绍了。另一种方法是通过软件设计来解决，例如在本例中，我们可以通过两次判断来确定硬币是否真的通过。具体方法是，在硬币通过时，首先判断出 P0 引脚的电压为高电平后，延时一段时间（这里设为 1 秒），然后再进行一次判断，如果 P0 引脚的状态发生改变，则变为低电平，说明硬币确实通过了。于是，将代码修改如下：

```
    coin_count=0 #定义存钱数
    pinflag=0    #定义引脚的标识位，用来判断是否短路

    while True:
    display.show(str(coin_count))#显示当前存钱数
    if (pin0.read_digital()==0 and pinflag==1):#判断两个金属夹片是否短路过，并且当前没有短路
        coin_count=coin_count+1       #存钱数加 1
        pinflag=0
    if(pin0.read_digital()==1):                #检查两个金属夹片是否短路
        pinflag=1
        sleep(1000)              #防抖动延时
```

如上面程序所示，在硬币通过时，首先进行一次判断，间隔 1 秒，再次进行判断。如果两次判断得到的硬币通过状态不同，则说明确实有硬币通过了。具体来看，当硬币投下时，第一次判断线路导通，pin0 为 1，间隔 1 秒后再次判断，由于硬币已落下，线路断开，所以 pin0 恢复为 0。如果出现这个过程，则表明硬币确实投入了，此时存钱数加 1。

10.4　代码分析

10.4.1　基本原理

存钱时从投币口投入硬币，硬币具有良好的导电性，会将两根导线连通，从而改变引脚的电平高低。当引脚电平为高时，说明两根导线连通，有硬币通过，从而进行数字的累加，并显示。为了防止硬币本身凹凸不平导致误判断，我们进行两次检测，第一次检测发现导线连通，经过 1 秒后，再检测一次，如果线路已断开，则说明硬币通过，数字加 1。

10.4.2　逻辑分析

（1）定义“变量”存钱数，用来计算当前存入的硬币数量。

（2）定义“引脚标识位”，用来判断硬币是否投入。

（3）在无限循环中，首先显示当前的存钱数，然后逻辑判断硬币是否投入。

（4）若发现硬币已经投入，则延迟 1 秒后，再进行一次判断，如果导线仍处于连通状态，则存钱数加 1。

【思考】

是否有其他判断硬币投入的方法呢？

10.5　知识要点

10.5.1　拼插编程

【掌握】

- TurnipBit 拼插编程网站的代码切换。
- 会使用变量。
- 会使用引脚。

【理解】

- 循环的应用。
- 循环在编程中的应用。
- 变量的定义。
- 引脚的使用及定义。
- 注释的灵活使用。

10.5.2 代码编程

【掌握】

- 变量的代码编写。
- 引脚的代码编写。

【理解】

循环主要命令的语法。

第 11 章　带小夜灯的电子时钟

11.1　制作带小夜灯的电子时钟

如果在床头放置一个带小夜灯的电子时钟，使用起来是不是很方便呢？白天，我们可以通过电子时钟知道时间；晚上，亮亮的电子时钟会影响我们休息，这时关掉电子时钟，小夜灯会自动亮起，是不是很酷？！如果夜晚想看时间怎么办？按下按键，时钟就会显示一次时间，很方便吧！上面描述的并不是什么创意性的产品，而是我们现在要做的带小夜灯的电子时钟，让你真正体会一次创意实现的乐趣。

所需器材：

- TurnipBit 开发板一块
- 下载数据线一条
- 废旧纸盒一个
- 导线若干
- DS3231 时钟模块一个
- 光敏电阻一个
- LED 灯一个
- 1kΩ 电阻一个
- 接入互联网的电脑一台（推荐使用 Google Chrome 或者 Firefox 浏览器）

11.2 基础知识

在制作电子时钟之前，我们先来认识一下电阻、光敏电阻、LED 和 TurnipBit 扩展板。

11.2.1 电阻

如果你学过物理的一些知识，那么对电阻这个概念就不会陌生。任何导体有电流通过时，都具有限流的特性。在物理学中，就是把导体对电流的阻碍作用叫电阻。电阻是导体的一种基本性质，与导体的尺寸、材料、温度有关。电阻用 *R* 表示。

1. 电阻分类

一般所说的电阻就是如图 11-1 所示的这种，其阻值不会发生变化。还有一种电阻，其阻值是可以调节的，称为可变电阻，又叫电位器。实际上，电阻按照不同的分类方法，可以分为很多种，如图 11-2 所示。本实验中使用的 1kΩ 电阻就属于具有固定阻值的电阻，光敏电阻属于特殊用途电阻。

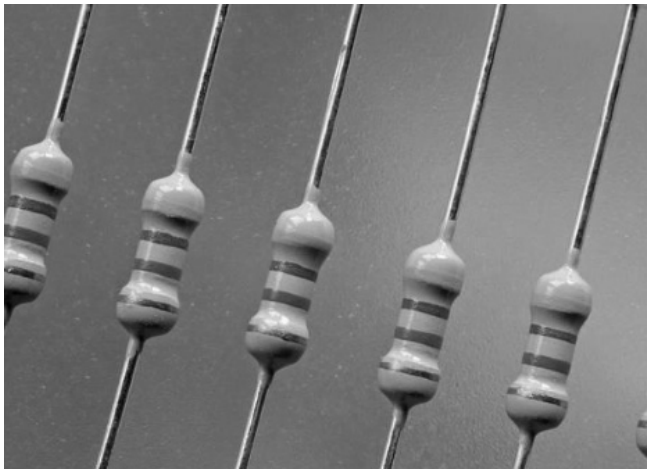


图 11-1 电阻

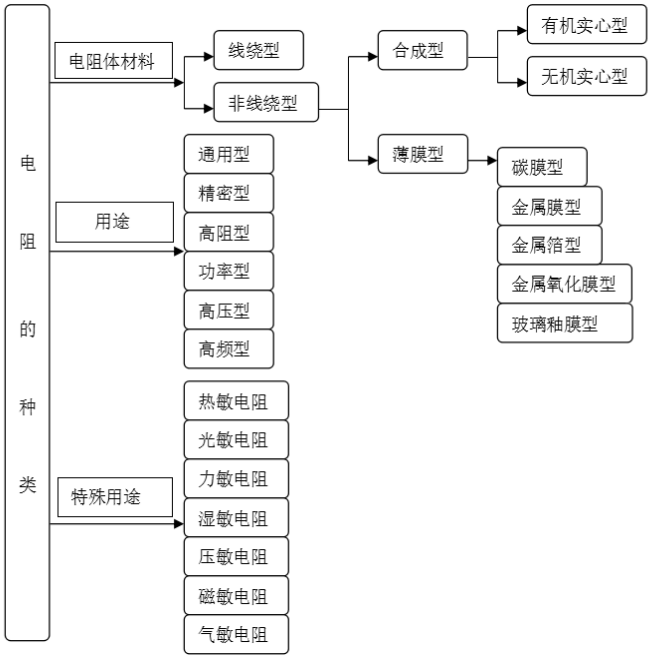


图 11-2　电阻分类

2．色环阻值标示法

色环电阻是指电阻上面用了四道色环（如图 11-3 所示）、五道色环（如图 11-4 所示）或者六道色环来表示电阻值，可以从任意角度一次性读取代表电阻值的颜色信息。色环电阻是应用于各种电子设备最多的电阻类型，无论怎样安装，维修者都能方便地读出其阻值，便于检测和更换。

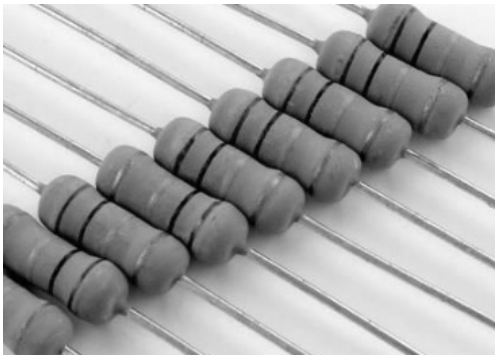


图 11-3　四道色环电阻

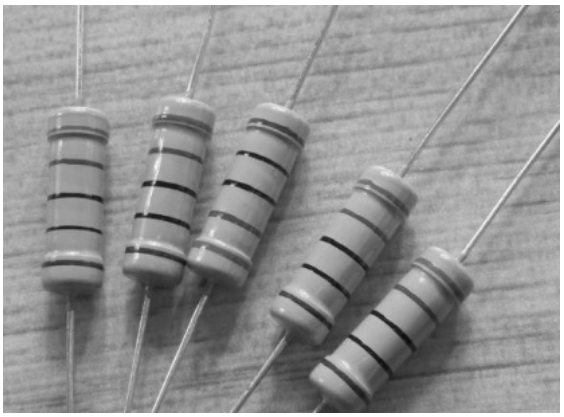
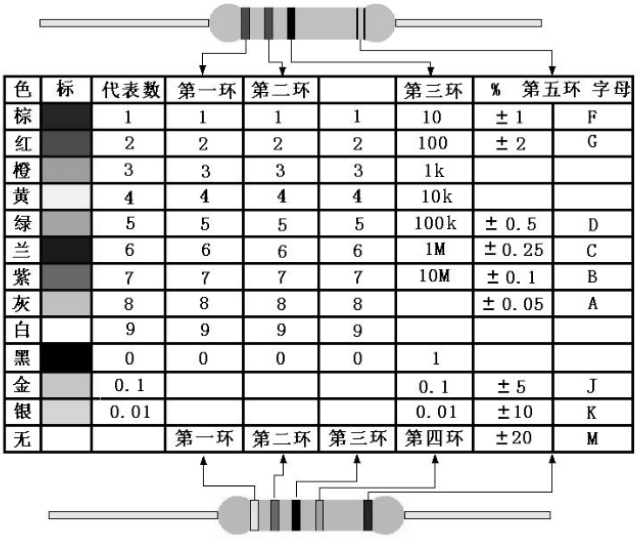


图 11-4　五道色环电阻

常见的色环阻值如图 11-5 所示。以四道色环电阻为例，单独在一边的一环为第四环，然后依次是第三环、第二环、第一环。第一环和第二环是数字环，第三环是倍率色环，第四环是误差环。在图 11-5 中，第一环为棕色表示 1，第二环为黑色表示 0，第三环为黄色表示 10kΩ，第四环为金色表示误差为 5%，则其阻值为 10kΩ×10 =100kΩ。



色	标	代表数	第一环	第二环		第三环	% 第五环 字母	
棕		1	1	1	1	10	±1	F
红		2	2	2	2	100	±2	G
橙		3	3	3	3	1k		
黄		4	4	4	4	10k		
绿		5	5	5	5	100k	±0.5	D
兰		6	6	6	6	1M	±0.25	C
紫		7	7	7	7	10M	±0.1	B
灰		8	8	8	8		±0.05	A
白		9	9	9	9			
黑		0	0	0	0	1		
金		0.1				0.1	±5	J
银		0.01				0.01	±10	K
无			第一环	第二环	第三环	第四环	±20	M

图 11-5　常见的色环阻值（单位：欧姆）

有时我们在识别电阻时会遇到一些不清晰的情况，这时就可以运用以下技巧来进行识别。

技巧 1：先找标志误差的色环，从而排定色环顺序。最常用的表示电阻误差的颜色是金色、银色和棕色，尤其是金色环和银色环，一般绝少用作电阻色环的第一环，所以在电阻上面只要有金色环和银色环，就可以基本认定这是色环电阻的最后一环。

技巧 2：棕色环是否是误差环标志的判别。棕色环既常用作误差环，又常作为有效数字环，且常常作为第一环和最后一环同时出现，使人很难识别谁是第一环。在实践中，可以按照色环之间的间隔加以判别，比如对于一个五道色环电阻而言，第五环和第四环之间的间隔比第一环和第二环之间的间隔要宽一些，据此可判定色环顺序。

技巧 3：在根据色环之间的间隔无法判定色环顺序的情况下，还可以利用电阻的生产序列值来加以判别。比如有一个电阻的色环读序是：棕、黑、黑、黄、棕，其值为 10kΩ×10=100kΩ，误差为 1%，属于正常的电阻值；若是反顺序读：棕、黄、黑、棕，其值为 14×1Ω=14Ω，误差为 1%。显然按照后一种排序所读出的电阻值，在电阻的生产序列中是没有的，故后一种色环顺序是不对的。

11.2.2　光敏电阻

科学家发现硫化镉、硒、硫化铝、硫化铅和硫化铋等材料在不同的光线照射下，具有电阻阻值发生规律性变化的特点，根据这一特点制作出了光敏电阻（见图 11-6）。光敏电阻器又称光导管，其特性是在特定光的照射下，它的阻值迅速减小。光敏电阻常用于检测可见光线，当光线变强时，其阻值减小；当光线变弱时，其阻值增大。光敏电阻器一般用于进行光的测量、光的控制和光电转换（将光的变化转换为电的变化）。在这个实验中，我们用光敏电阻来感知当前是否是熄灯睡觉的情况。在夜晚睡觉时，我们往往会关掉灯，这时光线最暗，光敏电阻的阻值会变大；相反，当天亮了时，光线变强，它的阻值会变小。

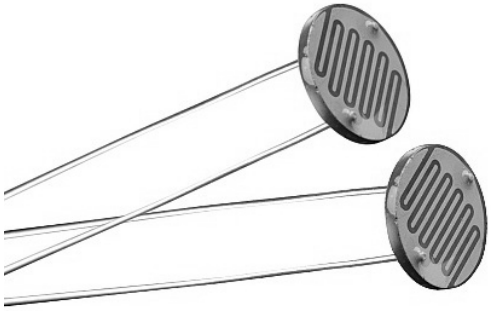


图 11-6　光敏电阻

11.2.3　LED

LED 又叫发光二极管（见图 11-7），是一种能够将电能转化为可见光的固态的半导体器件，它可以直接把电转化为光。LED 的心脏是一个半导体的晶片，晶片的一端附在一个支架上，其一端是负极，另一端连接电源的正极，使整个晶片被环氧树脂封装起来，就会发出光亮。LED 可以直接发出红色、黄色、蓝色、绿色、青色、橙色、紫色、白色的光。每个 LED 都有两个引脚，一长一短，短的连接 GND，长的连接正极。

图 11-7　LED（发光二极管）

11.2.4　TurnipBit 扩展板

TurnipBit 扩展板是用来将 TurnipBit 的引脚和主要功能脚引出的一个开发板（实物图如图 11-8 所示，示意图如图 11-9 所示）。利用 TurnipBit 扩展板，我们可以完成一些更加复杂的工作和任务。TurnipBit 扩展板包含以下几个区域。

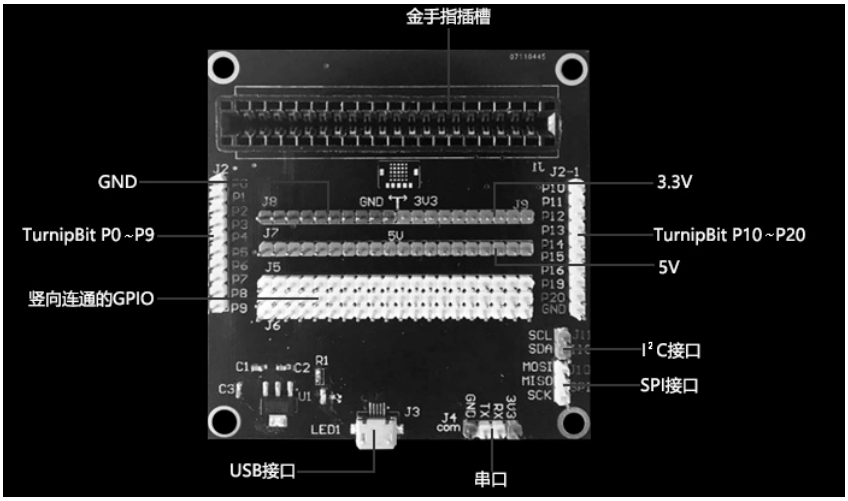


图 11-8　TurnipBit 扩展板实物图

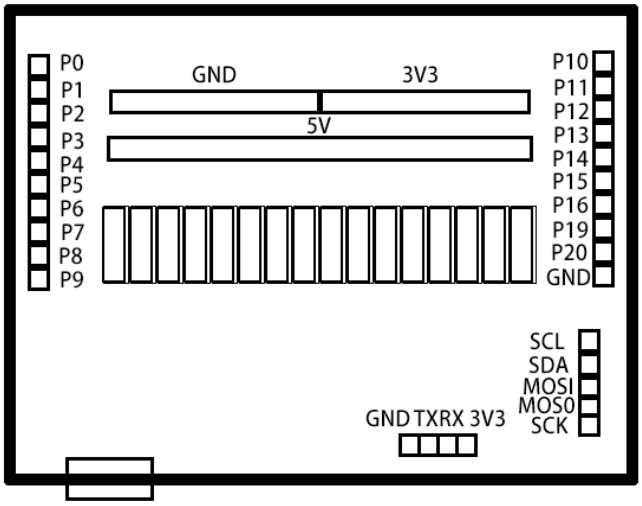


图 11-9　TurnipBit 扩展板示意图

- 金手指插槽：主要用于与 TurnipBit 开发板进行连接，正常插拔即可。
- 电源区：包括 GND、3.3V 和 5V 区域，提供电源供电。
- USB 接口：负责电源接入。
- GPIO 口：主要包括 P0~P20 的 GPIO 接口、I^2C 接口、串口以及 SPI 接口。

11.3 线路设计

本实验的主要目的是制作电子时钟和小夜灯，其中电子时钟主要通过 DS3231 模块来实现，使用 LED 和光敏电阻来实现小夜灯。

11.3.1 光敏电阻的使用

光敏电阻的使用方法有很多，这里我们只讲比较简单的一种方法。由于光敏电阻在不同的光线下其阻值会发生变化，根据这一特性，在电压相同的情况下，阻值发生变化后，分压的电压值就会不同。据此，我们可以将一个电阻与光敏电阻进行串联（电路示意图如图 11-10 所示），测量光敏电阻两端的电压，从而达到感知光线强弱的作用。

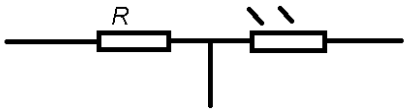


图 11-10　电路示意图

11.3.2 DS3231 的连接

DS3231（见图 11-11）是一个时钟模块，它自身带有电池，当设置好时间后，DS3231 就会自动走时。DS3231 与开发板之间的通信是通过 I^2C 接口来完成的。I^2C 总线是由 Philips 公司开发的一种简单的双向二线制同步串行总线，它只需要 SDA（串行数据线）和 SCL（串行时钟线）两根线即可在连接于总线的器件之间传送信息。SDA 和 SCL 都是双向 I/O 线，连接时，只需要将 DS3231 上的 SDA

和 SCL 与 TurnipBit 扩展板上的 SDA 和 SCL 相连即可。

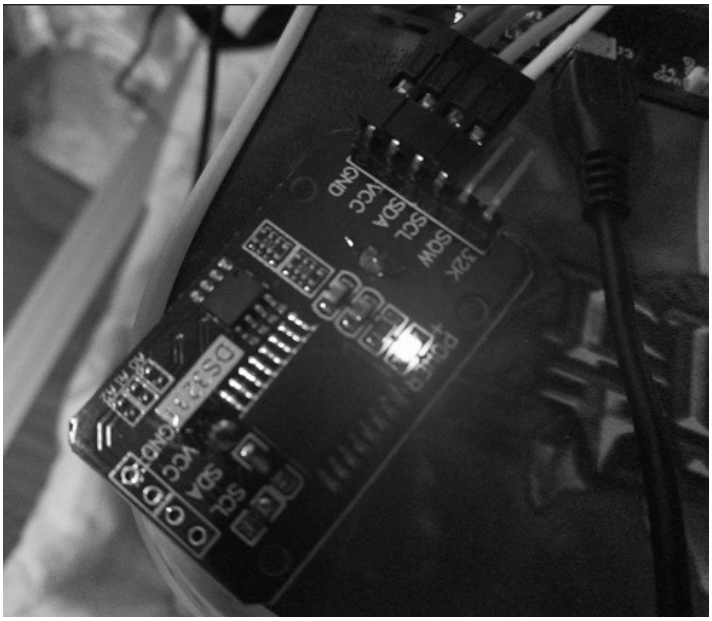


图 11-11　DS3231 实物图

11.3.3　器件的连接

本实验中需要连接的器件主要包括 LED、光敏电阻、电阻、DS3231，具体连接方法如下：

（1）LED 的长引脚（正极）与 TurnipBit 扩展板的 P1 相连，短引脚也就是负极与 GND 相连。我们只需要控制 P1 输出高电平还是低电平，就可以控制 LED 是亮还是灭。当 P1 输出高电平时，LED 的两个引脚形成回路，LED 亮起；当 P1 输出低电平时，LED 熄灭。

（2）光敏电阻与 1kΩ 电阻串联后，一端连接正极，一端连接负极，中间引出一条线，连接到 TurnipBit 扩展板的 P2 处。这里我们只需要读取 P2 引脚的模拟电压，就可以获取光线的变化情况。

（3）DS3231 与 TurnipBit 扩展板的连接如表 11-1 所示。

表 11-1　DS3231 与 TurnipBit 扩展板连线表

DS3231	TurnipBit 扩展板
GND	GND
VCC	3.3V
SCL	SCL
SDA	SDA

各器件连接实物图如图 11-12 所示。

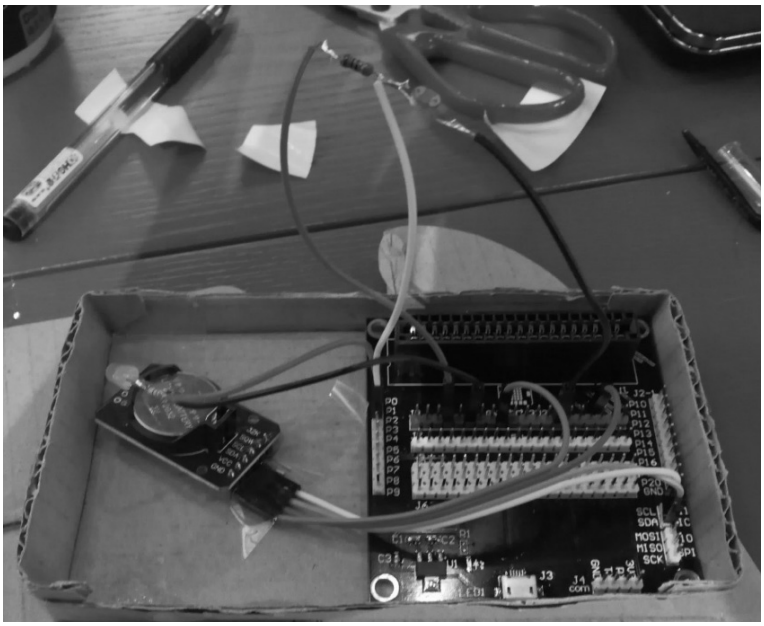


图 11-12　各器件连接实物图

11.4　程序设计

11.4.1　引脚的使用

在 TurnipBit 拼插编程中，专门有“引脚”块（如图 11-13 所示），其中包括“引脚 0 被触摸”“读取引脚 0 模拟电压”“设置模拟量 0 到引脚 0”“读取引脚 0 的数字输入”以及“设置数字输出 0 到引脚 0”。这里我们只使用“读取引脚 0 模

拟电压”和“设置数字输出 0 到引脚 0”。

图 11-13 “引脚”块

（1）“读取引脚 0 模拟电压”，主要用于读取光敏电阻线路中的模拟电压，通过不同的模拟电压来确定光线变化。比如读出引脚 P2 的值，则对应的代码为：

```
pin2.read_analog()
```

（2）“设置数字输出 0 到引脚 0”，主要用于设置引脚的值。比如对 P1 引脚进行设置，当设置数字输出 1 时，P1 引脚为高电平；当设置数字输出 0 时，P1 引脚为低电平。对应的代码为：

```
pin1.write_digital(1) #将 P1 引脚设置为 1
```

11.4.2　光敏电阻光线临界值的测量

不同的光敏电阻组成的线路对光线临界值的感应有时会有偏差，为了方便程序设计，首先需要找到光线临界值。根据 11.3 节介绍的线路设计，需要读取 P2 引脚的模拟电压，从而观察在光线全亮和全暗时读取到的值的变化（拼插如图 11-14 所示）。测试当光敏电阻的光线被完全遮住时（可以用手挡住来判断）所

获取的值，本实验中获取的值为 52，光线略微有一点时值为 54，于是选用 54 作为光线临界值。

图 11-14　读取引脚 2 模拟电压

11.4.3　DS3231 模块的代码

在 TurnipBit 中没有专门的 DS3231 时钟模块，需要我们自己构建一个模块。这里我们建立了一个 DS3231 类，通过调用类里的方法来实现对 DS3231 的控制。具体的类库代码如下：

```
DS3231_ADDR      = 0x68
DS3231_REG_SEC    = b'\x00'
DS3231_REG_MIN    = b'\x01'
DS3231_REG_HOUR   = b'\x02'
DS3231_REG_DAY    = b'\x04'
DS3231_REG_MONTH  = b'\x05'
DS3231_REG_YEAR   = b'\x06'
DS3231_REG_TEMP   = b'\x11'
class DS3231(object):

    def __init__(self):
        pass
    def DATE(self, dat=[]):
        if dat==[]:
            t = []
            t.append(self.year())
            t.append(self.month())
            t.append(self.day())
            return t
        else:
            self.year(dat[0])
            self.month(dat[1])
```

```
            self.day(dat[2])

    def TIME(self, dat=[]):
        if dat==[]:
            t = []
            t.append(self.hour())
            t.append(self.min())
            t.append(self.sec())
            return t
        else:
            self.hour(dat[0])
            self.min(dat[1])
            self.sec(dat[2])
    def DateTime(self, dat=[]):
        if dat==[]:
            return self.DATE() + self.TIME()
        else:
            self.year(dat[0])
            self.month(dat[1])
            self.day(dat[2])
            self.hour(dat[3])
            self.min(dat[4])
            self.sec(dat[5])

    def dec2hex(self, dat):
        return (int(dat/10)<<4) + (dat%10)

    def setREG(self, dat,reg):
        buf = bytearray(2)
        buf[0] = reg[0]
        buf[1] = dat
        i2c.write(DS3231_ADDR,buf, repeat=False)

    def getREG_DEC(self,reg):
        i2c.write(DS3231_ADDR,reg)
        t = i2c.read(DS3231_ADDR,1)[0]
        #print("--===",t,"----------")
        return (t>>4)*10 + (t%16)

    def sec(self, sec=''):
```

```
        if sec == '':
            return self.getREG_DEC(DS3231_REG_SEC)
        else:
            self.setREG(self.dec2hex(sec), DS3231_REG_SEC)

    def min(self, min=''):
        if min == '':
            return self.getREG_DEC(DS3231_REG_MIN)
        else:
            self.setREG(self.dec2hex(min), DS3231_REG_MIN)

    def hour(self, hour=''):
        if hour=='':
            return self.getREG_DEC(DS3231_REG_HOUR)
        else:
            self.setREG(self.dec2hex(hour), DS3231_REG_HOUR)

    def day(self, day=''):
        if day=='':
            return self.getREG_DEC(DS3231_REG_DAY)
        else:
            self.setREG(self.dec2hex(day), DS3231_REG_DAY)

    def month(self, month=''):
        if month=='':
            return self.getREG_DEC(DS3231_REG_MONTH)
        else:
            self.setREG(self.dec2hex(month), DS3231_REG_MONTH)

    def year(self, year=''):
        if year=='':
            return self.getREG_DEC(DS3231_REG_YEAR)
        else:
            self.setREG(self.dec2hex(year), DS3231_REG_YEAR)

    def TEMP(self):
        i2c.write(DS3231_ADDR,DS3231_REG_TEMP, repeat=False)
        t1 = i2c.read(DS3231_ADDR,1, repeat=False)[0]
        i2c.write(DS3231_ADDR,b'\x12', repeat=False)
        t2 = i2c.read(DS3231_ADDR,1, repeat=False)[0]
```

```
        if t1>0x7F:
            return t1 - t2/256 -256
        else:
            return t1 + t2/256
```

如果看不懂上面的代码也没关系，你只需要搞清楚下面几个方法就可以了。

（1）DATE()，用于获取 DS3231 时钟日期，返回一个包括年、月、日的数组，如 2017 年 9 月 10 日，返回[17,9,10]。

（2）TIME()，用于获取 DS3231 时钟时间，返回一个包括时、分、秒的数组，如 12:23:32，返回[12,23,32]。

（3）DATE([yy,mm,dd])，用于给 DS3231 设定时钟日期。

（4）TIME([hh,mm,ss])，用于给 DS3231 设定时钟时间。

在调用以上方法时，往往有两种方式，其中一种方式是直接用类名调用，例如：

```
Time=DS3231().TIME()
```

另一种方式是先将 DS3231()赋值给一个变量，然后用变量来进行引用，例如：

```
ds=DS3231()
Time=ds.TIME()
```

11.4.4　时钟对时代码

在本实验中，我们没有在按键上加调时的功能，因此需要用代码为 DS3231 进行一次时钟对时。完成时钟对时后，我们只需要从 DS3231 模块中读取时间就可以了。假设当前时间为 2017 年 10 月 29 日 14 点 54 分 00 秒，时钟对时的具体代码如下：

```
from microbit import *

DS3231_ADDR      = 0x68
DS3231_REG_SEC    = b'\x00'
DS3231_REG_MIN    = b'\x01'
DS3231_REG_HOUR   = b'\x02'
DS3231_REG_DAY    = b'\x04'
DS3231_REG_MONTH  = b'\x05'
```

```
DS3231_REG_YEAR   = b'\x06'
DS3231_REG_TEMP   = b'\x11'
class DS3231(object):

    def __init__(self):
        pass
    def DATE(self, dat=[]):
        if dat==[]:
            t = []
            t.append(self.year())
            t.append(self.month())
            t.append(self.day())
            return t
        else:
            self.year(dat[0])
            self.month(dat[1])
            self.day(dat[2])

    def TIME(self, dat=[]):
        if dat==[]:
            t = []
            t.append(self.hour())
            t.append(self.min())
            t.append(self.sec())
            return t
        else:
            self.hour(dat[0])
            self.min(dat[1])
            self.sec(dat[2])
    def DateTime(self, dat=[]):
        if dat==[]:
            return self.DATE() + self.TIME()
        else:
            self.year(dat[0])
            self.month(dat[1])
            self.day(dat[2])
            self.hour(dat[3])
            self.min(dat[4])
            self.sec(dat[5])
```

```
    def dec2hex(self, dat):
        return (int(dat/10)<<4) + (dat%10)

    def setREG(self, dat,reg):
        buf = bytearray(2)
        buf[0] = reg[0]
        buf[1] = dat
        i2c.write(DS3231_ADDR,buf, repeat=False)

    def getREG_DEC(self,reg):
        i2c.write(DS3231_ADDR,reg)
        t = i2c.read(DS3231_ADDR,1)[0]
        #print("--===",t,"----------")
        return (t>>4)*10 + (t%16)

    def sec(self, sec=''):
        if sec == '':
            return self.getREG_DEC(DS3231_REG_SEC)
        else:
            self.setREG(self.dec2hex(sec), DS3231_REG_SEC)

    def min(self, min=''):
        if min == '':
            return self.getREG_DEC(DS3231_REG_MIN)
        else:
            self.setREG(self.dec2hex(min), DS3231_REG_MIN)

    def hour(self, hour=''):
        if hour=='':
            return self.getREG_DEC(DS3231_REG_HOUR)
        else:
            self.setREG(self.dec2hex(hour), DS3231_REG_HOUR)

    def day(self, day=''):
        if day=='':
            return self.getREG_DEC(DS3231_REG_DAY)
        else:
            self.setREG(self.dec2hex(day), DS3231_REG_DAY)

    def month(self, month=''):
```

```
        if month=='':
            return self.getREG_DEC(DS3231_REG_MONTH)
        else:
            self.setREG(self.dec2hex(month), DS3231_REG_MONTH)

    def year(self, year=''):
        if year=='':
            return self.getREG_DEC(DS3231_REG_YEAR)
        else:
            self.setREG(self.dec2hex(year), DS3231_REG_YEAR)

    def TEMP(self):
        i2c.write(DS3231_ADDR,DS3231_REG_TEMP, repeat=False)
        t1 = i2c.read(DS3231_ADDR,1, repeat=False)[0]
        i2c.write(DS3231_ADDR,b'\x12', repeat=False)
        t2 = i2c.read(DS3231_ADDR,1, repeat=False)[0]
        if t1>0x7F:
            return t1 - t2/256 -256
        else:
            return t1 + t2/256

ds=DS3231()
ds.DATE([17,10,29])  #设定日期
ds.TIME([14,54,00])  #设定时间
while True:
    Date=ds.DATE()
    Time=ds.TIME()
    Time_s='20'
    for i in Date:
        Time_s+=str(i)+'-'
    Time_s=Time_s[:-1]+' '
    for i in Time:
        Time_s+=str(i)+':'
    display.scroll(Time_s[:-1]) #显示时间，查看是否设定成功
    sleep(1000)
```

将以上代码复制到 TurnipBit 的编程区，保存 HEX 文件并下载到 TURNIPBIT 磁盘后，程序就会运行，如果 TurnipBit 显示的时间为所设定的时间，则说明设定成功。

11.4.5　带小夜灯的电子时钟的代码实现

下面我们用代码来实现带小夜灯的电子时钟。

1．伪代码

根据带小夜灯的电子时钟的实际需求，本程序的伪代码为：

① 读取时钟。

② 读取光敏电阻的模拟电压，通过读取 5 次计算平均值的方法来判断光线的强弱。如果平均值大于或等于 54，则说明当前光线较强；如果平均值小于 54，则说明光线较弱，打开 LED 灯，关闭时钟显示。

③ 在关闭时钟显示的情况下，判断按键曾经按下的次数，如果按键曾经按下的次数大于 0，则说明在以上过程中有按键按下过，此时显示时钟时间。

④ 回到第 1 步，重新循环。

2．具体代码

具体代码如下：

```
from microbit import *

DS3231_ADDR      = 0x68
DS3231_REG_SEC    = b'\x00'
DS3231_REG_MIN    = b'\x01'
DS3231_REG_HOUR   = b'\x02'
DS3231_REG_DAY    = b'\x04'
DS3231_REG_MONTH  = b'\x05'
DS3231_REG_YEAR   = b'\x06'
DS3231_REG_TEMP   = b'\x11'
class DS3231(object):

    def __init__(self):
        pass
    def DATE(self, dat=[]):
        if dat==[]:
            t = []
            t.append(self.year())
```

```
            t.append(self.month())
            t.append(self.day())
            return t
        else:
            self.year(dat[0])
            self.month(dat[1])
            self.day(dat[2])

    def TIME(self, dat=[]):
        if dat==[]:
            t = []
            t.append(self.hour())
            t.append(self.min())
            t.append(self.sec())
            return t
        else:
            self.hour(dat[0])
            self.min(dat[1])
            self.sec(dat[2])
    def DateTime(self, dat=[]):
        if dat==[]:
            return self.DATE() + self.TIME()
        else:
            self.year(dat[0])
            self.month(dat[1])
            self.day(dat[2])
            self.hour(dat[3])
            self.min(dat[4])
            self.sec(dat[5])

    def dec2hex(self, dat):
        return (int(dat/10)<<4) + (dat%10)

    def setREG(self, dat,reg):
        buf = bytearray(2)
        buf[0] = reg[0]
        buf[1] = dat
        i2c.write(DS3231_ADDR,buf, repeat=False)

    def getREG_DEC(self,reg):
```

```
        i2c.write(DS3231_ADDR,reg)
        t = i2c.read(DS3231_ADDR,1)[0]
        #print("--===",t,"----------")
        return (t>>4)*10 + (t%16)

    def sec(self, sec=''):
        if sec == '':
            return self.getREG_DEC(DS3231_REG_SEC)
        else:
            self.setREG(self.dec2hex(sec), DS3231_REG_SEC)

    def min(self, min=''):
        if min == '':
            return self.getREG_DEC(DS3231_REG_MIN)
        else:
            self.setREG(self.dec2hex(min), DS3231_REG_MIN)

    def hour(self, hour=''):
        if hour=='':
            return self.getREG_DEC(DS3231_REG_HOUR)
        else:
            self.setREG(self.dec2hex(hour), DS3231_REG_HOUR)

    def day(self, day=''):
        if day=='':
            return self.getREG_DEC(DS3231_REG_DAY)
        else:
            self.setREG(self.dec2hex(day), DS3231_REG_DAY)

    def month(self, month=''):
        if month=='':
            return self.getREG_DEC(DS3231_REG_MONTH)
        else:
            self.setREG(self.dec2hex(month), DS3231_REG_MONTH)

    def year(self, year=''):
        if year=='':
            return self.getREG_DEC(DS3231_REG_YEAR)
        else:
            self.setREG(self.dec2hex(year), DS3231_REG_YEAR)
```

```
    def TEMP(self):
        i2c.write(DS3231_ADDR,DS3231_REG_TEMP, repeat=False)
        t1 = i2c.read(DS3231_ADDR,1, repeat=False)[0]
        i2c.write(DS3231_ADDR,b'\x12', repeat=False)
        t2 = i2c.read(DS3231_ADDR,1, repeat=False)[0]
        if t1>0x7F:
            return t1 - t2/256 -256
        else:
            return t1 + t2/256

ds=DS3231()

while True:
    Date=ds.DATE()
    Time=ds.TIME()
    Time_s='20'
    for i in Date:
        Time_s+=str(i)+'-'
    Time_s=Time_s[:-1]+' '
    for i in Time:
        Time_s+=str(i)+':'

    hour=Time[0]
    sumnum=0

    for j in range(0,5):   #计算 5 次电压平均值

        sumnum=sumnum+pin2.read_analog()
    averagenum=sumnum/5
    if averagenum < 54:   #如果小于 54 则说明光线较弱
        pin1.write_digital(1)   #打开 LED 灯
    else:                 #如果大于或等于 54，则说明光线较强
        pin1.write_digital(0)   #关闭 LED 灯

        display.scroll(Time_s[:-1])   #显示时钟时间
    if button_a.get_presses() > 0 or button_b.get_presses() > 0:
        display.scroll(Time_s[:-1])  #如果按过按键，则显示时钟时间
    sleep(1000)
```

将以上代码复制到 TurnipBit 的编程区，保存 HEX 文件并下载到 TURNIPBIT

磁盘后，我们会发现 TurnipBit 已经开始工作了。

【思考】

试着根据上面的代码绘制流程图。

11.5　外壳组装

完成了硬件和软件设计以后，下一步就是进行外壳的组装。外壳的设计有很多方法，如果你有 3D 打印的条件，则完全可以自己设计一个 3D 外壳的模型，然后用 3D 打印机打印一个外壳。如果没有这个条件也没关系，现在我们就用纸盒来做一个外壳。首先需要找到一个纸盒，然后将 TurnipBit 开发板及各模块固定在纸盒上，如图 11-15 所示。

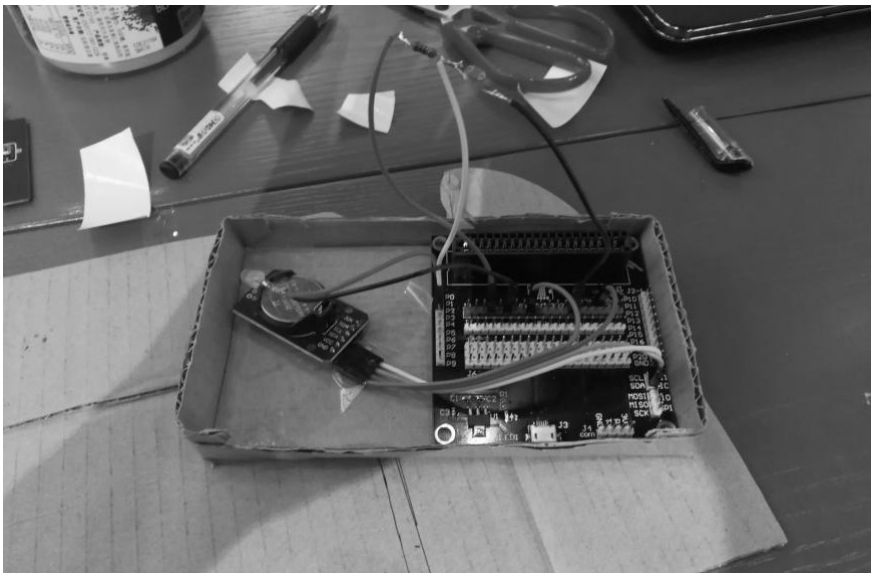


图 11-15　将 TurnipBit 开发板及各模块固定在纸盒上

再找来一个相对软的外盒，抠出 LED 灯的口、光敏电阻的孔及 TurnipBit 的孔，然后把做好的纸盒塞进去，从而形成一个简易的外壳。最后，带着小夜灯的电子时钟效果图如图 11-16 所示。

图 11-16　带小夜灯的电子时钟效果图

http://v.youku.com/v_show/id_XMzE1MTQ1MDY2NA==.html?spm=a2hzp.8244740.0.0 这里提供了视频效果，你可以在浏览器中进行查看。

11.6　知识要点

11.6.1　拼插编程

【掌握】

- 什么是 LED。
- 什么是光敏电阻。
- DS3231 模块。

【理解】

引脚的使用。

11.6.2　代码编程

【掌握】

引脚的读取和设置。

【理解】

I^2C 接口的使用。

第 12 章　会思考的避障车

12.1　什么是会思考的避障车

在日常生活中，我们经常会看到各种各样的遥控车，它们需要我们去操作，控制其前进、后退和转弯。今天就带大家认识一个不一样的新朋友——“会思考的避障车”。“会思考的避障车”和遥控车最主要的区别就是智能化，它不需要我们去操控，自己就能行走。同时它还会实时检测前方是否有障碍物，思考自己是否要前进或者转弯。看到这里，你是不是已经跃跃欲试了？话不多说，动起手来吧！

所需器材：

- TurnipBit 开发板一块
- 下载数据线一条
- 智能小车套件一套（底盘、车轮、电机等）
- 超声波模块（HC-SR04）一个（用作小车的“眼睛”）
- L298N 电机驱动模块一个
- 接入互联网的电脑一台（推荐使用 Google Chrome 或者 Firefox 浏览器）

12.2　基础知识

12.2.1　电机

1．电机的概念

电机（俗称“马达”）是指根据电磁感应定律实现电能转换或传递的一种电磁装置。电机在电路中用字母 M 表示，其主要作用是产生驱动转矩，作为电器或各种机械的动力源。发电机在电路中用字母 G 表示，其主要作用是将电能转化为机械能。

2．电机的分类

电机有很多种，常见的分类方式如下：

- 按工作电源种类划分，可分为直流电机和交流电机。
- 按结构和工作原理划分，可分为直流电机、异步电机和同步电机。
- 按启动与运行方式划分，可分为电容启动式单相异步电机、电容运转式单相异步电机、电容启动运转式单相异步电机和分相式单相异步电机。
- 按用途划分，可分为驱动用电机和控制用电机。
- 按转子的结构划分，可分为笼型感应电机（旧标准称为“鼠笼型异步电机”）和绕线转子感应电机（旧标准称为“绕线型异步电机”）。
- 按运转速度划分，可分为高速电机、低速电机、恒速电机和调速电机。

本章中，我们重点介绍常用的直流电机和步进电机。

3．直流电机的概念及原理

直流电机是指能将直流电能转化成机械能（直流电机）或将机械能转化成直流电能（直流发电机）的旋转电机（见图 12-1）。当它作为电机运行时是直流电机，将电能转化为机械能；当它作为发电机运行时是直流发电机，将机械能转化为电能。

图 12-1　直流电机

直流电机的结构由定子和转子两大部分组成。直流电机运行时静止不动的部分称为定子，其主要作用是产生磁场，由机座、主磁极、换向极、端盖、轴承和电刷装置等组成。运行时转动的部分称为转子，其主要作用是产生电磁转矩和感应电动势，是直流电机进行能量转换的枢纽，所以通常又称为电枢，由转轴、电枢铁心、电枢绕组、换向器和风扇等组成。直流电机有正负两个引脚，只要一个引脚接电源正极，一个引脚接电源负责，电机就会旋转工作。如果转换两个引脚的电源极性，电机就会反向旋转。

4．步进电机的概念及原理

与直流电机不同，步进电机是指能将电脉冲信号转变为角位移或线位移的开环控制电机（如图 12-2 所示）。在非超载的情况下，电机的转速、停止的位置只取决于脉冲信号的频率和脉冲数，而不受负载变化的影响。当步进驱动器接收到一个脉冲信号时，它就驱动步进电机按设定的方向转动一个固定的角度，称为“步距角”，它的旋转是以固定的角度一步一步进行的。可以通过控制脉冲个数来控制角位移量，从而达到准确定位的目的；同时可以通过控制脉冲频率来控制电机转动的速度和加速度，从而达到调速的目的。

步进电机是一种可以自由回转的电磁铁，其动作原理是依靠气隙磁导的变化来产生电磁转矩。通常步进电机的转子为永磁体，当电流流过定子绕组时，定子绕组产生一个矢量磁场，该磁场会带动转子旋转一个角度，使得转子的磁场方向与定子的磁场方向一致。当定子的矢量磁场旋转一个角度时，转子也随着该磁场

转动一个角度。每输入一个电脉冲，电机就转动一个角度前进一步。它输出的角位移与输入的脉冲数成正比，转速与脉冲频率成正比。改变绕组通电的顺序，电机就会反转。所以可通过控制脉冲数量、频率及电机各相绕组的通电顺序来控制步进电机的转动。

图 12-2　步进电机

5．电机驱动模块

当使用单片机控制直流电机时需要增加驱动电路，用来提供足够的电流。H 桥驱动电路是直流电机驱动电路中比较常见的一种电路，它主要实现直流电机正反两个方向的转动。

如图 12-3 所示是简单的 H 桥驱动电路，其形状类似于字母“H”，作为负载的直流电机像“桥”一样架在上面，所以称为“H 桥驱动”，4 个开关所在的位置就称为“桥臂”。从图中可以看出，当开关 A、D 接通时，直流电机正向转动；而当开关 B、C 接通时，直流电机将反向转动，从而实现了电机的正反控制。

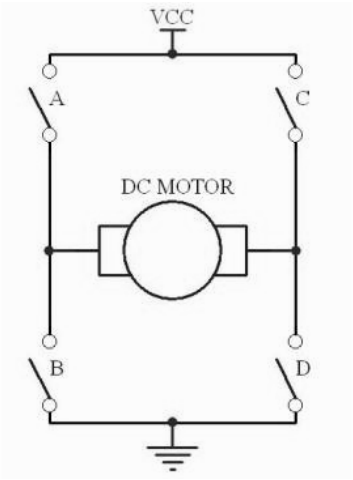


图 12-3　简单的 H 桥驱动电路

本实验中将使用 L298N 驱动模块，如图 12-4 所示。该模块是 ST 公司生产的一种高电压、大电流的电机驱动芯片，是比较典型的 H 桥驱动电路的体现，可以同时驱动两台直流电机或一台两相步进电机和四相步进电机。L298N 的主要特点如下：

- 工作电压高，最高工作电压可达 46V。
- 输出电流大，瞬间峰值可达 3A，持续工作电流为 2A。

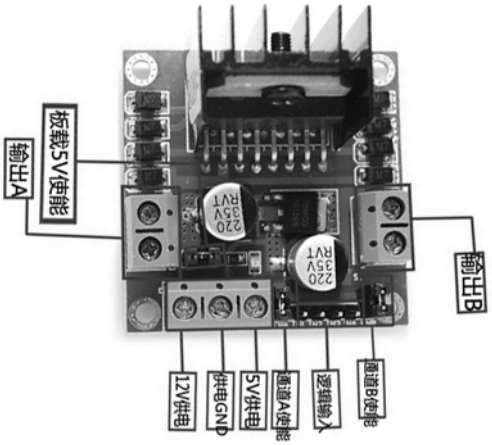


图 12-4　L298N 驱动模块

12.2.2　超声波传感器

超声波传感器是将超声波信号转换成其他能量信号（通常是电信号）的传感器（如图 12-5 所示）。超声波是振动频率高于 20kHz 的机械波，具有频率高、波长短、绕射现象小等特点，特别是方向性好，能够成为射线而定向传播。超声波对液体、固体的穿透力很大，尤其对不透明的固体。超声波碰到杂质或分界面时会产生显著反射形成回波，碰到活动物体时能产生多普勒效应。

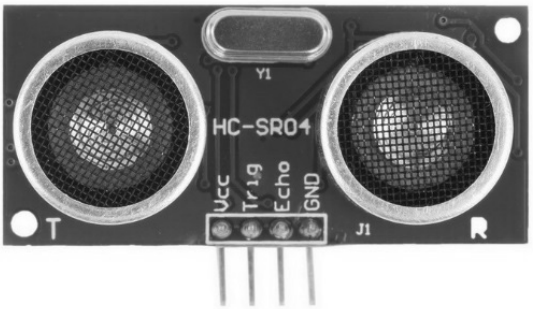


图 12-5　超声波传感器

超声波传感器主要通过发送超声波并接收超声波来对某些参数或事项进行检测。发送超声波由发送器部分完成，主要利用振子的振动产生并向空中辐射超声波；接收超声波由接收器部分完成，主要接收由发送器辐射出的超声波并将其转换为电能输出。发送器与接收器的动作都受控制部分控制，如控制发送器发出超声波的脉冲频率、占空比、探测距离等。

本实验使用的超声波模块是 HC-SR04，其工作原理如下：

- IO 口 Trig 引脚用来触发测距，需要提供至少 10μs 的高电平。
- 触发测距后，超声波模块自动发送 8 个 40kHz 的方波，自动检测是否有信号返回。
- 当超声波模块接收到信号返回后，通过 IO 口将 Echo 引脚置为高电平，高电平持续的时间就是超声波从发送到返回的时间，即测试距离=（高电平持续的时间×声速（340m/s）/2）。

12.3 避障车的组装

12.3.1 硬件器件

避障车的硬件组成主要包括以下几个部分。

（1）底盘一片，如图 12-6 所示。

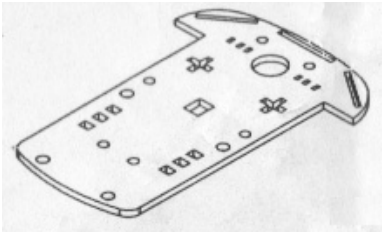


图 12-6 底盘

（2）轮子两个，如图 12-7 所示。

图 12-7 轮子

（3）测速码盘两片，如图 12-8 所示。

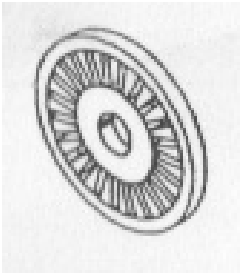


图 12-8 测速码盘

（4）减速直流电机两台，如图 12-9 所示。

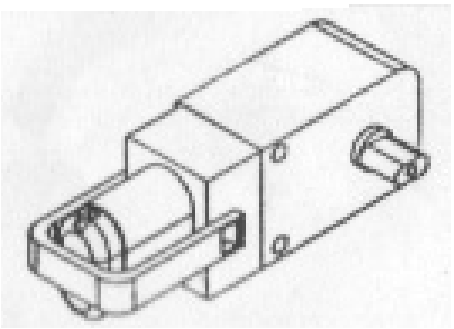


图 12-9　减速直流电机

（5）4 节 5 号电池盒一个，如图 12-10 所示。

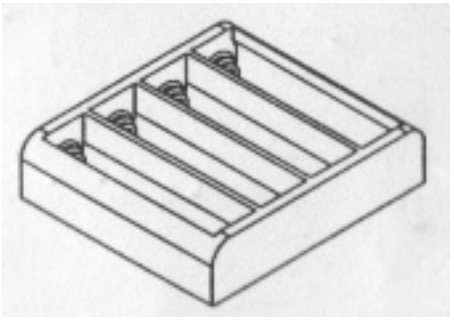


图 12-10　5 号电池盒

（6）万向轮一个，如图 12-11 所示。

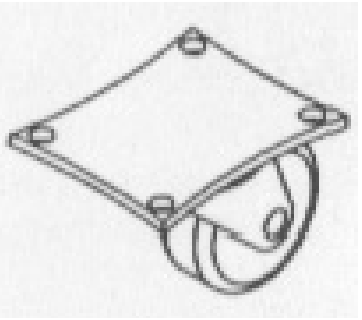


图 12-11　万向轮

（7）M3×30 螺丝 4 个，如图 12-12 所示。

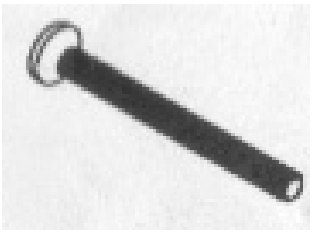


图 12-12　M3×30 螺丝

（8）船形开关一个，如图 12-13 所示。

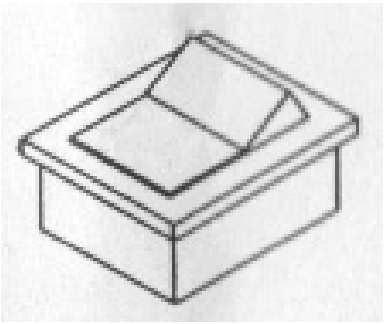


图 12-13　船形开关

（9）紧固件 4 片，如图 12-14 所示。

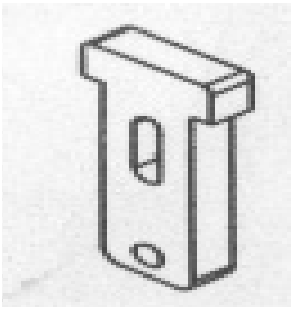


图 12-14　紧固件

（10）M3×6 螺丝 8 个，如图 12-15 所示。

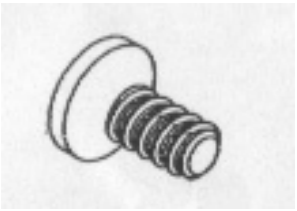


图 12-15　M3×6 螺丝

（11）M3 螺帽 8 个，如图 12-16 所示。

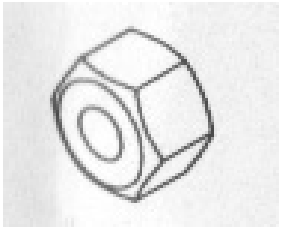


图 12-16　M3 螺帽

（12）L12 铜柱 4 个，如图 12-17 所示。

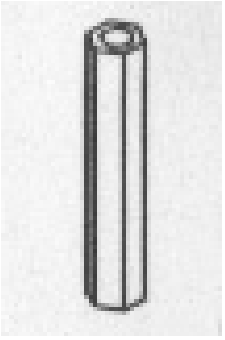


图 12-17　L12 铜柱

12.3.2　硬件安装步骤

① 先把小车底盘、紧固件、码盘的黄色保护纸撕掉，然后把紧固件插在小车底盘上，如图 12-18 所示。

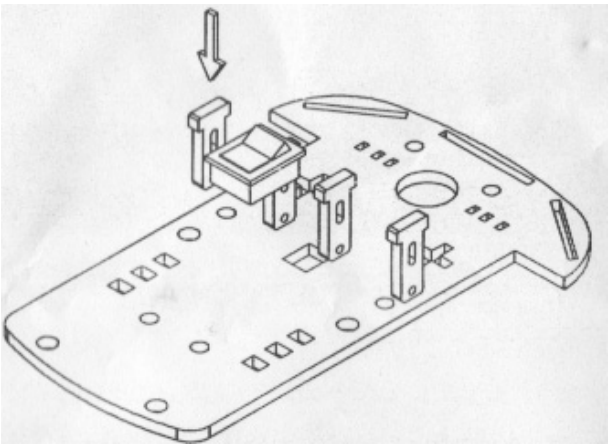


图 12-18　连接底盘与紧固件

② 安装码盘，并把电机固定在底盘上，如图 12-19 所示。码盘轴心一面大一面小，大的一面往电机轴上插（注意：电机引线铜片朝向内侧，即码盘一端）。

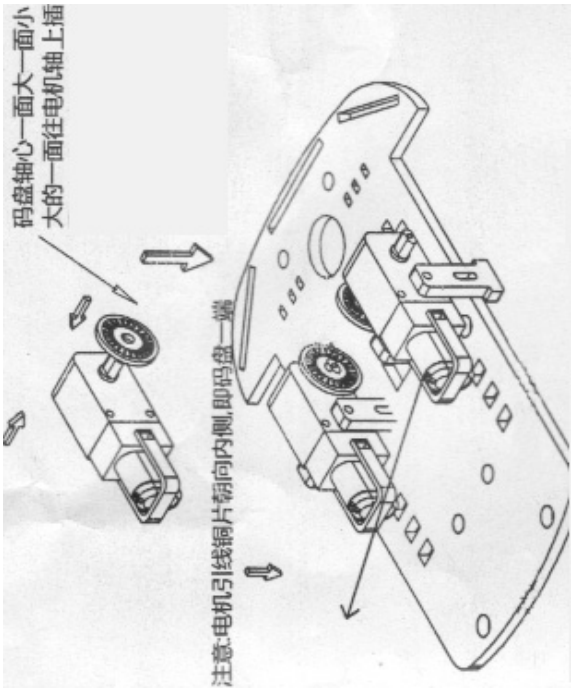


图 12-19　安装码盘

③ 插入螺丝，把电机固定到小车底盘上，并拧上螺帽，如图 12-20 所示。

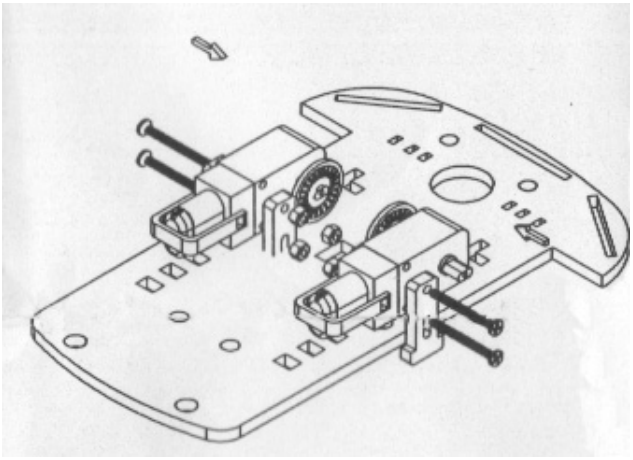


图 12-20　固定电机

④ 插入螺丝，固定电池盒，如图 12-21 所示。本实验中这一步可以省略，我们使用充电宝进行供电。

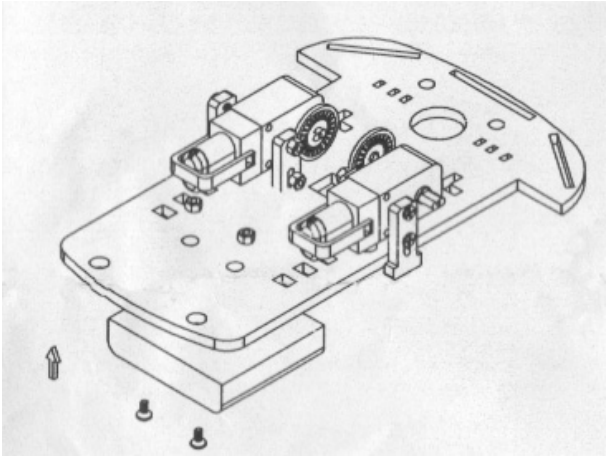


图 12-21　固定电池盒

⑤ 放入铜柱，拧紧 8 个螺丝固定万向轮，如图 12-22 所示。然后手捏住电机（保护紧固件），并往里面插入轮子，组装完成。

最后，我们来看一下组装效果图，如图 12-23 所示。

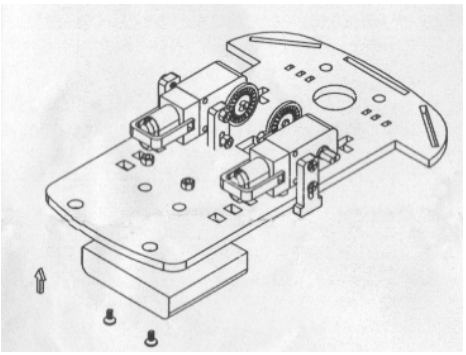


图 12-22　固定万向轮

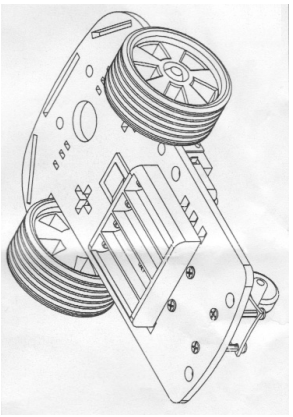


图 12-23　组装效果图

12.3.3　电机驱动模块和超声波模块的安装

硬件的连接，主要是各引脚的连接，具体如表 12-1 所示。

表 12-1　引脚连接表

TurnipBit 扩展板	超声波模块	L298N
5V	VCC	+5V
GND	GND	GND
P5	Trig	
P8	Echo	
P11		IN4
P12		IN3

续表

TurnipBit 扩展板	超声波模块	L298N
P13		IN2
P19		IN1

超声波模块共有 4 个引脚，分别是 VCC、GND、Trig 和 Echo。其中 VCC 接 TurnipBit 扩展板的 5V 引脚，GND 接 TurnipBit 扩展板的 GND 引脚，Trig（触发引脚）接 P5 引脚，Echo（回传引脚）接 P8 引脚。电机驱动模块 L298N 左侧的 OUT3、OUT4 接线端子对应接入左轮电机的下侧、上侧铜片，右侧也是如此。L298N 的输入端 IN1、IN2、IN3、IN4 分别接入 TurnipBit 扩展板的 P19、P13、P12、P11 引脚。本实验利用充电宝进行供电，只需通过 USB 线将充电宝连接到 TurnipBit 开发板的 microusb 口即可。各器件连接示意图如图 12-24 所示。避障车组装图如图 12-25 所示。充电宝供电图如图 12-26 所示。

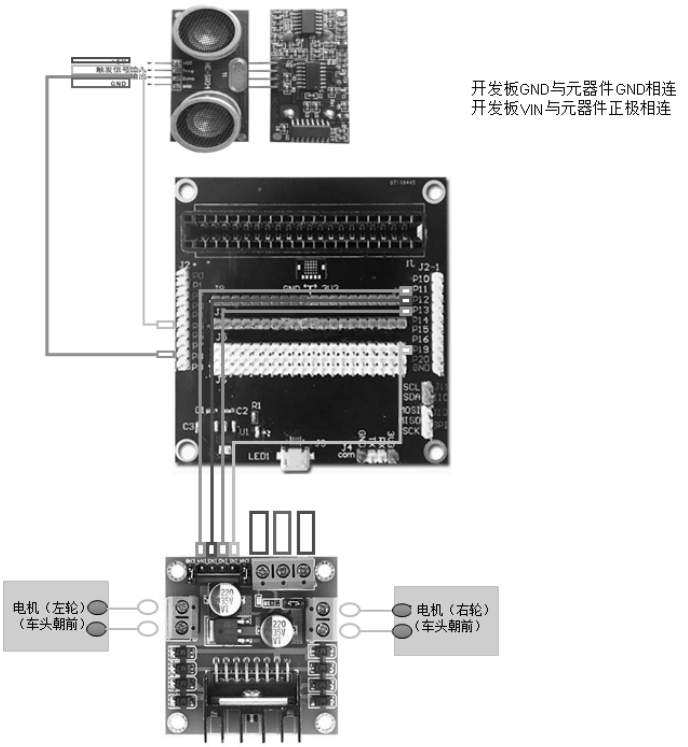


图 12-24　各器件连接示意图

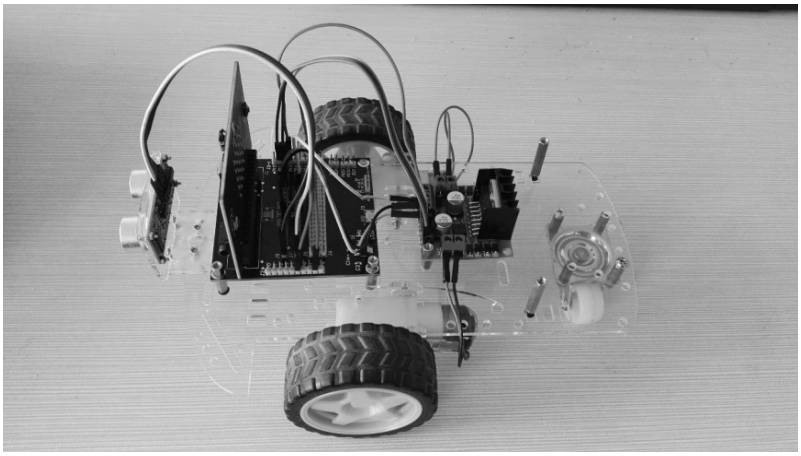


图 12-25　避障车组装图

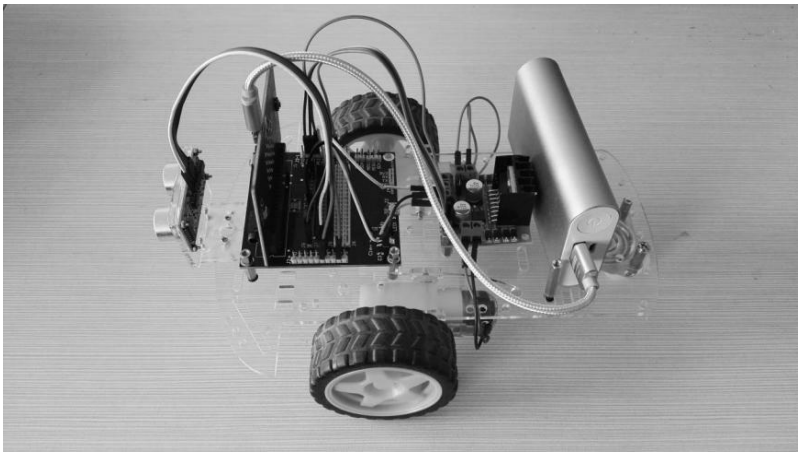


图 12-26　充电宝供电图

12.4　程序设计

12.4.1　伪代码分析

我们先用伪代码来分析一下避障车的程序逻辑。首先通过避障车的超声波模块来检测前方是否存在障碍物，若存在障碍物，则计算出与障碍物之间的距离。

然后判断当与障碍物之间的距离小于或等于我们预设的安全距离时，控制小车进行转向，避开障碍物；当与障碍物之间的距离大于我们预设的安全距离时，控制小车继续前进。逻辑顺序如下：

① 超声波模块开始检测。

② 计算出与前方障碍物之间的距离。

③ 判断与障碍物之间的距离是否小于或等于所设定的安全距离。

④ 若小于或等于所设定的安全距离，则进行转向，避开障碍物。

⑤ 若大于所设定的安全距离，则继续前进。

⑥ 从第 1 步开始重复。

根据伪代码画出流程图，如图 12-27 所示。

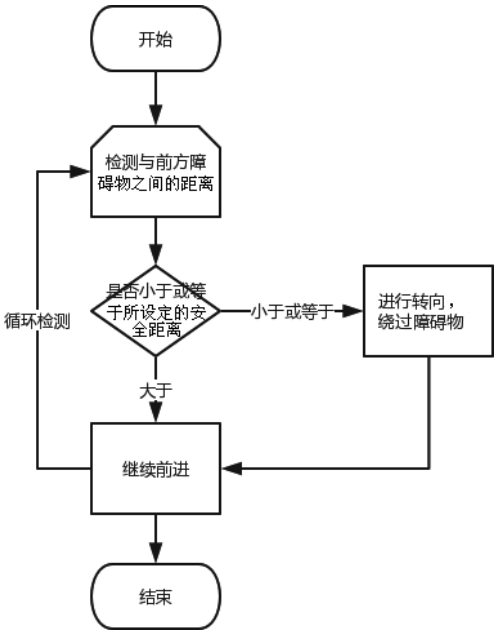


图 12-27　避障车的流程图

12.4.2　拼插编程

① 因为 P11 和 P12 引脚控制一个车轮，P13 和 P19 引脚控制另一个车轮，只

需要给相应车轮的 P12 和 P19 引脚高电平，另外两个引脚低高平，避障车的两个车轮就会转动。如果此时两个轮子的转动方向不同或者是向后转动的，则说明你在连线时没有注意连接的 L298N 引脚，只需要调换一下，避障车就会向前走，如图 12-28 所示。

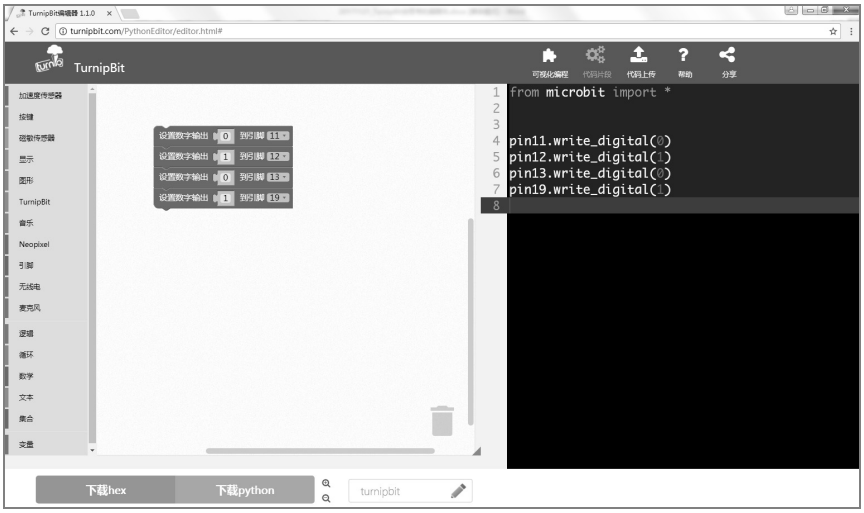


图 12-28　控制前进

② 添加一个无限循环，使程序一直运行，如图 12-29 所示。

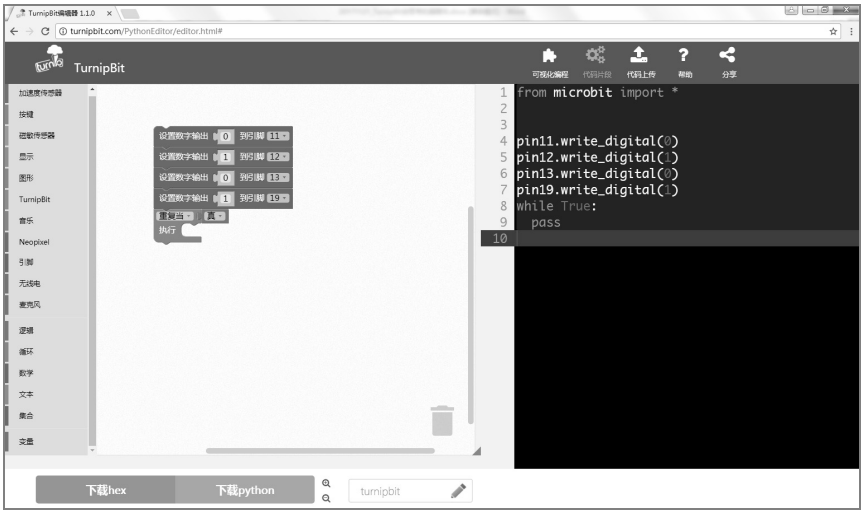


图 12-29　持续前进

③ 接下来需要完成检测障碍物的功能，这部分需要超声波模块来实现。我们需要在无限循环内添加检测障碍物的内容，让避障车一直不断地检测与前方障碍物之间的距离，如图 12-30 所示。

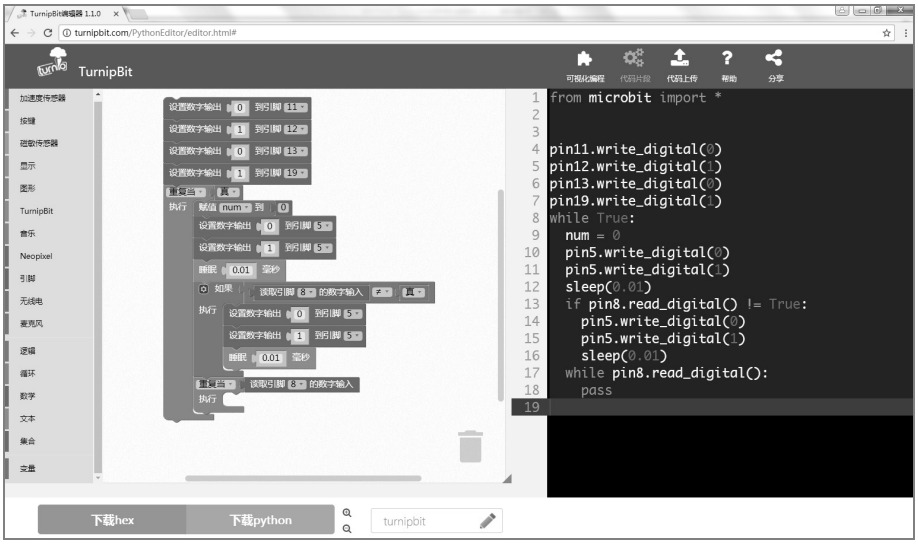


图 12-30　超声波测距

④ 为了更简单、方便、快捷地实现功能，我们采用计数的方式，测量一个大概的距离，然后判定是否进行避障。细心的同学可能会发现，在程序一开始的时候定义了一个 num 变量，我们就用 num 来计数。当引脚 8 为高电平时，表示超声波模块的 Echo 引脚进入接收模式，不断地让 num 自加 1，直到超声波模块接收到返回信号，即引脚 8 为低电平时，停止计数。这里的 num 就能反映出当前避障车与障碍物之间的距离。这里以 25 厘米为安全距离进行测试，发现 num 的值为 8。这说明，如果避障车与障碍物之间的距离小于 25 厘米，num 就小于 8。所以，在程序中判断如果 num 的值小于或等于 8，就进行转向避障，如图 12-31 所示。

⑤ 至此，会思考的避障车就完成了。为了让避障车更加炫酷，我们可以使用 LED 屏来动态显示当前避障车行驶的方向。方法是在避障车做出转向动作前，先用箭头来显示转向的方向，然后再做动作，如图 12-32 所示。

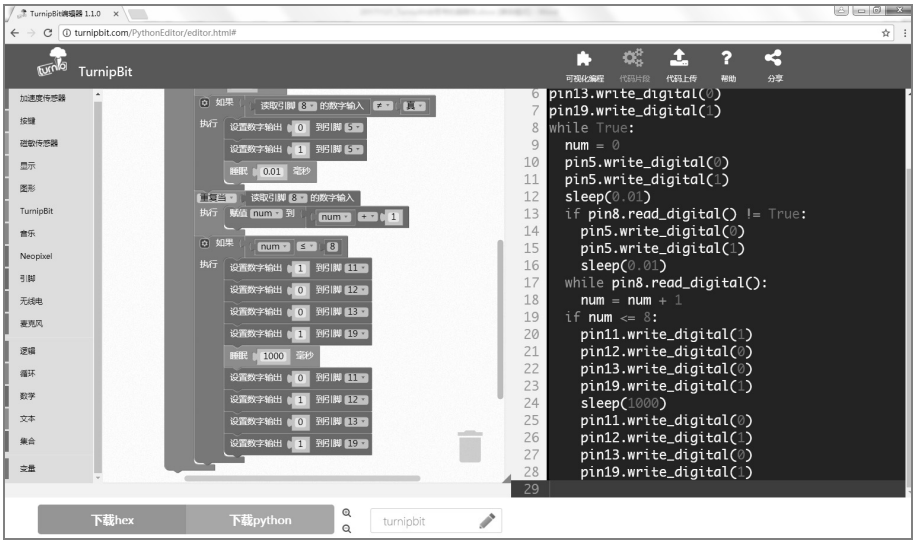


图 12-31　转向判断

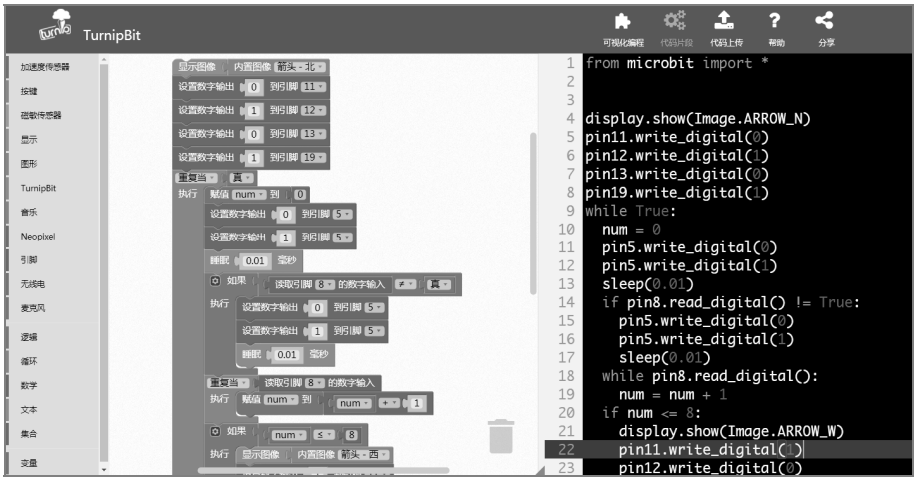


图 12-32　加入转向方向箭头

⑥ 将程序名修改成 turnipbit-car，点击“下载 hex”按钮将程序保存到电脑里，如图 12-33 所示。把所保存的 turnipbit- car.hex 文件拖入 TURNIPBIT 磁盘中，我们会看到 TurnipBit 板子上的灯在闪烁，说明正在下载到控制板中。下载成功后，将 TurnipBit 正确插入 TurnipBit 扩展板的金手指卡槽内，然后就启动避障车吧。

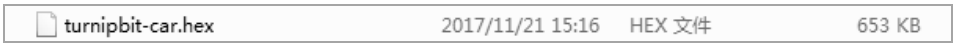


图 12-33　保存文件

12.4.3　代码分析

会思考的避障车的整体代码如下：

```
from microbit import *

display.show(Image.ARROW_N)
pin11.write_digital(0)
pin12.write_digital(1)
pin13.write_digital(0)
pin19.write_digital(1)
while True:
  num = 0
  pin5.write_digital(0)
  pin5.write_digital(1)
  sleep(0.01)
  if pin8.read_digital() != True:
    pin5.write_digital(0)
    pin5.write_digital(1)
    sleep(0.01)
  while pin8.read_digital():
    num = num + 1
  if num <= 8:
    display.show(Image.ARROW_W)
    pin11.write_digital(1)
    pin12.write_digital(0)
    pin13.write_digital(0)
    pin19.write_digital(1)
    sleep(1000)
    display.show(Image.ARROW_N)
    pin11.write_digital(0)
    pin12.write_digital(1)
    pin13.write_digital(0)
    pin19.write_digital(1)
```

接下来，我们一起来分析代码。从整体代码来看，控制避障车前进的代码我们编写了两遍，这在实际项目中是不合理的。为了让代码更加简洁、优雅，我们将控制避障车前进的代码提取出来建立一个 Go()函数，在需要执行控制前进代码的地方直接调用 Go()函数即可。

```
from microbit import *

def Go():
    display.show(Image.ARROW_N)
    pin11.write_digital(0)
    pin12.write_digital(1)
    pin13.write_digital(0)
    pin19.write_digital(1)

Go()
while True:
    num = 0
    pin5.write_digital(0)
    pin5.write_digital(1)
    sleep(0.01)
    if pin8.read_digital() != True:
        pin5.write_digital(0)
        pin5.write_digital(1)
        sleep(0.01)
    while pin8.read_digital():
        num = num + 1
    if num <= 8:
        display.show(Image.ARROW_W)
        pin11.write_digital(1)
        pin12.write_digital(0)
        pin13.write_digital(0)
        pin19.write_digital(1)
        sleep(1000)
        Go()
```

按照同样的方式，我们将控制转向和检测障碍物的代码提取出来分别建立 Turn()和 Detection()函数。

```
from microbit import *
```

```
def Go():
    display.show(Image.ARROW_N)
    pin11.write_digital(0)
    pin12.write_digital(1)
    pin13.write_digital(0)
    pin19.write_digital(1)

def Turn():
    display.show(Image.ARROW_W)
    pin11.write_digital(1)
    pin12.write_digital(0)
    pin13.write_digital(0)
    pin19.write_digital(1)

def Detection():
    num=0
    pin5.write_digital(0)
    pin5.write_digital(1)
    sleep(0.01)
    if pin8.read_digital() != True:
        pin5.write_digital(0)
        pin5.write_digital(1)
        sleep(0.01)
    while pin8.read_digital():
        num = num + 1
    return num

Go()
while True:
    num = Detection()
    if num <= 8:
        Turn()
        sleep(1000)
        Go()
```

【思考】

如何制作一个无线遥控车？

12.5 知识要点

12.5.1 拼插编程

【掌握】

- 电机的基本知识。
- 直流电机和步进电机。
- 超声波检测的原理。

【理解】

- 电机驱动电路的作用。
- L298N 电机驱动模块。

12.5.2 代码编程

【掌握】

引脚的读取和设置。

【理解】

代码编写的技巧和优化。